TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TAUNO TOIKKA

FEASIBILITY OF SELECTED MACHINE LEARNING METHODS FOR FAILURE FORECASTING OF AEROPLANE FLIGHT CONTROL SURFACES

Master of Science thesis

Examiner: Prof. Kari T. Koskinen
Examiner and topic approved by the
Faculty Council of the Faculty of
Department of Mechanical Engineering
and Industrial Systems
on 17th March 2017

ABSTRACT

TAUNO TOIKKA: Feasibility of selected machine learning methods for failure forecasting of aeroplane flight control surfaces
Tampere University of Technology
Master of Science thesis, 61 pages, 2 Appendix pages
17th March 2017
Master's Degree Programme Mechanical Engineering and Industrial Systems
Major: Analysis of Machines and Structures
Examiner: Prof. Kari T. Koskinen
Keywords: Machine Learning, Failure prediction, Failure forecasting, Condition monitoring, Self-Organizing Map, Support Vector Machine, Neural Network, Radial Basis Function, K-mean clustering, aeroplane flight control surface.

In this study the feasibility of some common machine learning algorithms such as Self Organizing Map (SOM), Support Vector Machine (SVM), Neural Network (NN), Radial Basis Function (RBF) and K-mean clustering for detecting the upcoming failure of the aeroplane flight control surfaces was studied. The machine learning algorithms were tested by the flight data from several similar type aeroplanes. The study was twofold. In the first part the research question was: Which samples of the historical data of properly working system are indicating the upcoming failure? In the second part the research question was: How to detect these failure indicating data samples from the new data? In the first part SOM and K-mean clustering showed a great applicability for detecting anomalies from the data before the actual occurrence of the failure. This result was further used to define which data samples were indicating the upcoming failure and what to further teach to supervised learning machine. In the first part SOM showed a great potential for detecting anomalies from healthy historical data and in this way helped to find failure indicators. In the second part SVM and NN showed a great capability of classifying *failure indicating healthy data samples* (FIHDS) out of the new data of properly working system. The sudden and significant increase of FIHDS's in system data indicated a correctly the upcoming failure.

TIIVISTELMÄ

TAUNO TOIKKA: Koneoppimismenetelmien Soveltuvuus Lentokoneen Siiven Ohjainpintojen Vikaantumisten Ennustamiseen
Tampereen teknillinen yliopisto
Diplomityö, 61 sivua, 2 liitesivua
17 Maaliskuu 2017
Konetekniikan DI-tutkinto-ohjelma
Pääaine: Koneiden ja rakenteiden analysointi
Tarkastajat: Prof. Kari T. Koskinen
Avainsanat: Koneoppiminen, Itseorganisoituva kartta, Tukivektorikone, Neurovekko, Lentokoneen siiven ohjauspinta, Vikaantumisen ennustaminen

Tässä työssä tutkittiin miten tietyt yleiset koneoppimismenetelmät kuten SOM, SVM, NN, RBF ja K-mean clustering soveltuvat lentokoneen siiven ohjauspintojen vikojen ennustamiseen. Kyseiset koneoppimismenetelmät testattiin useista lentokoneista tallennetulla lentodatalla. Tämä tutkimus oli kaksi osainen. Ensimmäisessä osassa selvitettiin mitkä datanäytteet toimivassa systeemissä indikoivat tulevaa vikaa. Toisessa osassa tutkittiin sitä miten näitä vikaa indikoivia datanäytteitä pystytään tunnistamaan uudesta datasta. Ensimmäisessä osassa SOM ja K-mean clustering menetelmien avulla pystyttiin löytämään poikkeavuuksia datasta jo ennen varsinaista vikaantumishetkeä. Toisessa osassa SVM:n ja NN:n avulla pystyttiin erottelemaan vikaa indikoivia datanäytteitä uudesta datasta.

PREFACE

This study was mainly financed by Finnish Defence Forces Logistics Command and hereby I would like to humbly thank this direction for financing this study and supplying the material for this study and in this way to make this whole research process possible. Here I would also like to thank my wife Iida for being supportive through the whole study process and my half year old son Valto for making sure that I would not oversleep. I would also like to deeply thank my superior Jouko Laitinen for the inspiring ideas and constructive criticism. My thanks are also devoted to professor Kari T. Koskinen for being my supervisor and helping to finish this work.

Tampere, 17.8.2016

Tauno Toikka

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

BMU	Best Matching Unit
PHDS	Pure Healthy Data Sample
FDFLC	Finnish Defence Forces Logistics Command
FIHDS	Failure Indicating Healthy Data Sample
NN	Neural Network
RBF	Radial Basis Function
SOM	Self-Organizing Map
SVM	Support Vector Machine
VC dimension	Vapnik-Chervonenkis dimension

1. INTRODUCTION

Failure forecasting is a challenging task. However when well established it can provide valuable information for example to as a support to condition based maintenance task decisions. Failure forecasting can be done by many ways such as by human observations followed making an intuitive conclusions or by monitoring the system data and interpret the data by some data analysis methods. The subject of this study focus on the data analysis side.

In generally the data based predictions about the system behaviour can be done by firs producing some model describing the system and then using the data on the model. The model of the system can be analytical/knowledge-baser model or black box model. By black box here is meant the system having the input and output but no real world related interpretations available for the system parameters. The model selection here was driven by the fact well stated in Dreyfus G [10] "...knowledge-based model requires that a theory be available, whereas the design of a black-box model requires that measurements be available."

The failure process of the application of this study is complex and no analytical theory for solving the problem is available. On the other hand the system has such a character that a lot of measured data about the system is available. Thus the black box model is chosen here for the prediction model. Machine learning methods user here are presenting black box methods.

Machine learning requires data. The data is the raw material for the machine learning and there is no learning without the data. However the quality of the data can vary a lot, and thus may add some extra challenges on machine learning. In the case of system failure forecasting the optimal data would be the the data monitored specially for the purpose. In optimal case the system would be censored in consistent way such that for example in places which could overheat before the failure there will be temperature censors, in places where the vibration level might increase before the failure there would be vibration censors, and so on.

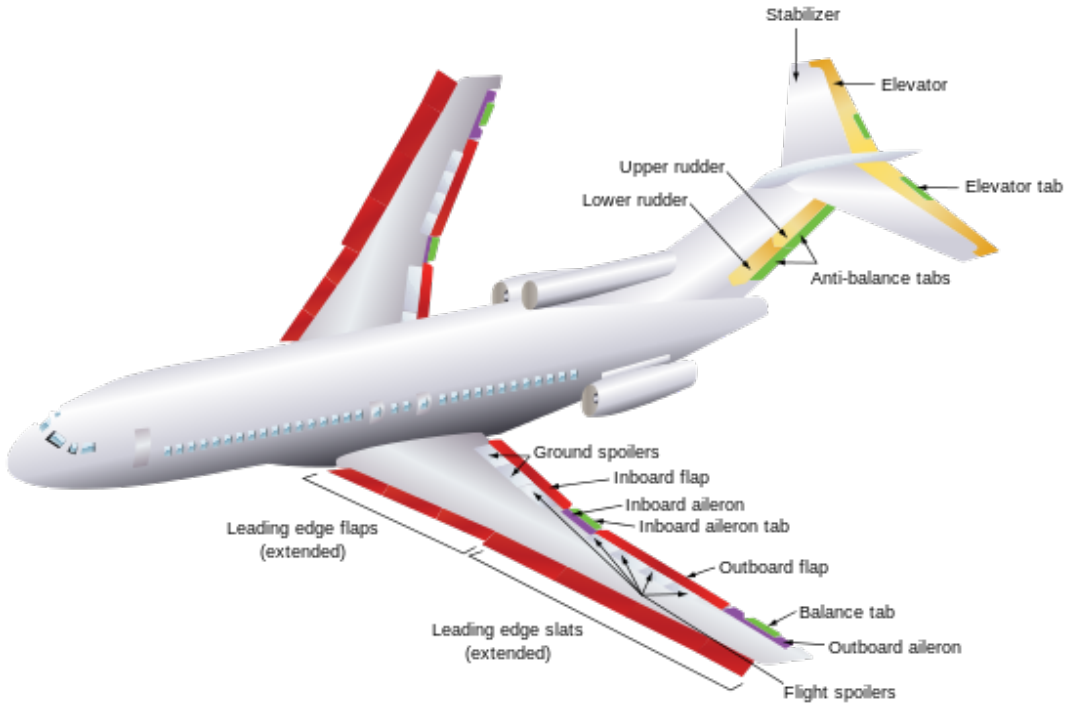


Figure 1.1 *The illustration of aeroplane flight control surfaces [2].*

The quality of the data available for this study is challenging since it is not the data designed for the condition monitoring but the data is just some arbitrary process data of the system. The data is from the aeroplanes of the Finnish Air Forces and the failure cases under the study are the occasionally failing aeroplane flight control surfaces (see figure 1.1).

One aeroplane from which the data is from have been flight several flights with properly working flight control surfaces. At one flight the control surface did seized. Before the next flight control surface was fixed. During the next flight the control surface did seized again. Since the failure of flight control surfaces may in some cases be critical, it is relevant to know could the failure have been able to predict beforehand.

Here any measurements specific for monitoring the condition of control surfaces was not available but only some arbitrary flight data from which only few parameters were directly related with control surfaces. Thus the comprehensive research question here is: "Could the future system failure be predicted beforehand on-line

based on the history of data of properly working runs and one failure run when data available is not specially monitored for the purpose?"

The dataset available for this study is described in the table 1.1. One dataset is in size scale of $1000-1000000$ samples encapsulating several tens measured parameters during the flight. *Healthy flights* in table 1.1 denotes the flights which performed properly working. *Failure flights* on the other hand denotes the flights which started as a properly working but ended up failing.

Table 1.1 *Datasets from separate flights available for this study.*

	Healthy flights	**Failure flights**
Aeroplane A	40	2
Aeroplane B	1	0
Aeroplane C	1	0
Aeroplane D	1	0

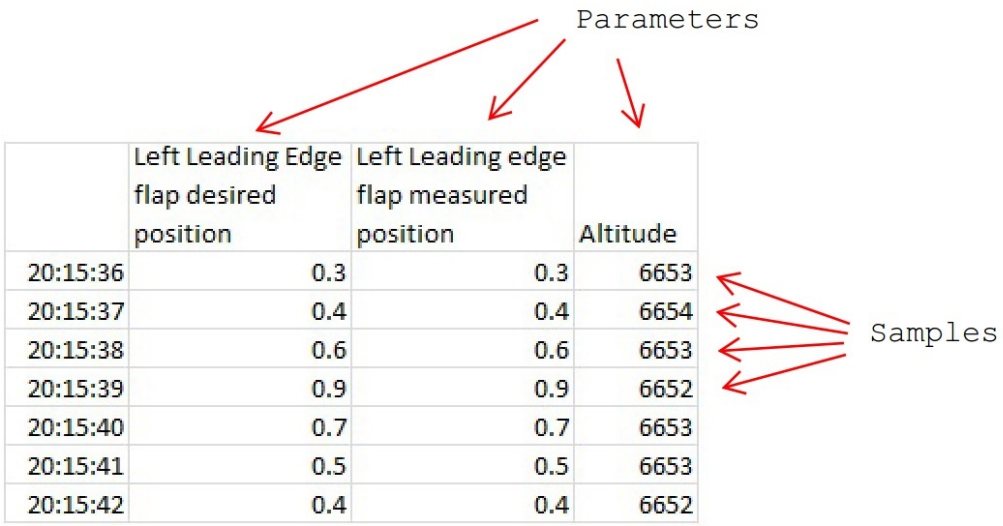
The Aeroplane A is under special investigation here. This aeroplane did flew 40 healthy flights, then some of its control surfaces did seized, then the surfaces were fixed and during the next flight the seizure did occur again. In this study it was examined could the second seizure have been predicted beforehand based on the data presented in the table 1.1.

1.1 Description of the data

From here on the seizure of the control surface is just denoted as failure. In this study the terms *Samples* and *Parameters* of the data will be used a lot and the meaning of these terms are described in figure 1.2. Also terms *healthy data sample* and *failure data sample* are in key role in this study and they will be defined next.

Healthy data sample

Healthy data sample here is the data sample with all its parameters from the moment when the control surface actual position corresponds with the desired position. Figure 1.3 illustrates the set of sequential healthy data samples. Since the measured position corresponds in error range with the desired position in has been defined here that this dataset is having only healthy data samples. When comparing figure 1.3 to



	Left Leading Edge flap desired position	Left Leading edge flap measured position	Altitude
20:15:36	0.3	0.3	6653
20:15:37	0.4	0.4	6654
20:15:38	0.6	0.6	6653
20:15:39	0.9	0.9	6652
20:15:40	0.7	0.7	6653
20:15:41	0.5	0.5	6653
20:15:42	0.4	0.4	6652

Figure 1.2 *Description of the data samples and parameters of some hypothetical data set.*

figure 1.4 it more obvious why the samples in figure 1.3 are denoted as an healthy samples. Healthy data samples are divided here into two subclasses: *Pure Healthy Data Sample* (PHDS) and *Failure Indicating Healthy Data Sample* (FIHDS).

Pure Healthy Data Sample (PHDS) is the healthy data sample that does not possess any information about the upcoming failure.

Failure Indicating Healthy Data Sample (FIHDS) is a healthy data sample which possess some information about the upcoming failure.

Failure data sample

Failure data sample is the data sample with all its parameters from the moment when the control surface actual position does not corresponds with the desired position (see figure figure 1.4). Failure data samples are practically not used in this study in any other way than for deciding of which flights will end with failure and which not.

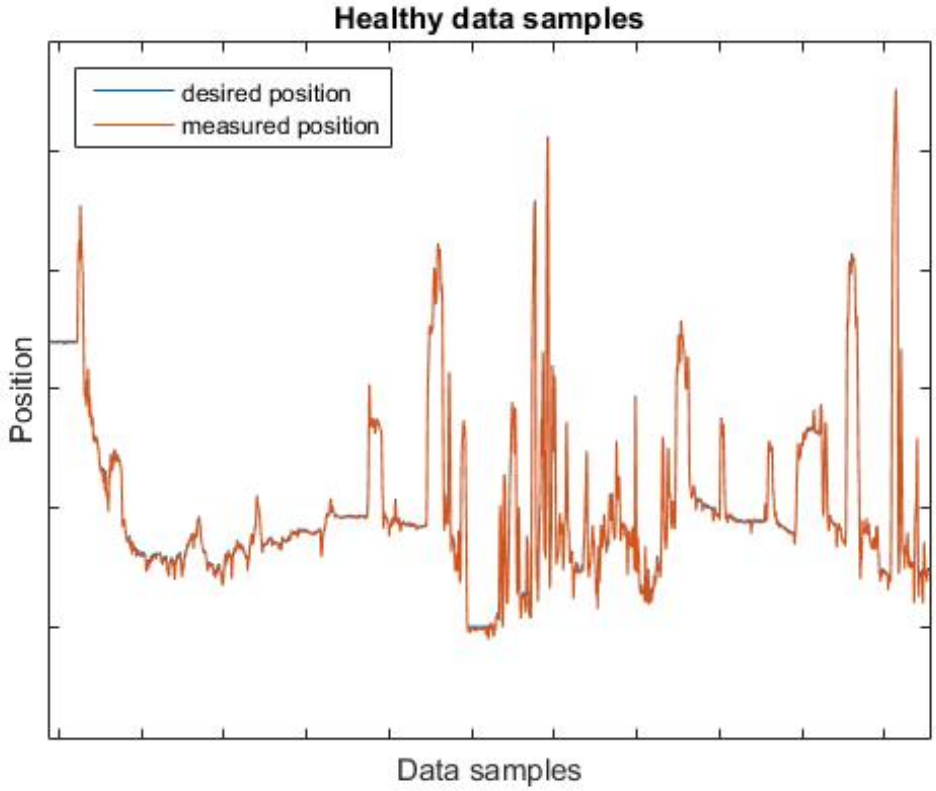


Figure 1.3 *Illustration of healthy data. Axis values are anonymized.*

1.2 Structure of the study

The study is twofold and thus can be divided in two separate parts. The first part (Part 1) focus on deciding which healthy data samples before the failure are FIHDS's. This is not an obvious task since for example by looking figure 1.4 how do we know which samples of the healthy data samples are FIHDS's? All of them? Part of them? None of them? Are the samples in the figure 1.3 also FIHDS's?

In order to solve the problem three methods have been used:

1. Self-Organizing Maps (SOM)

2. K-means clustering

3. Heuristic approach

The second part (Part 2) of this focus on *supervised learning machines* and training

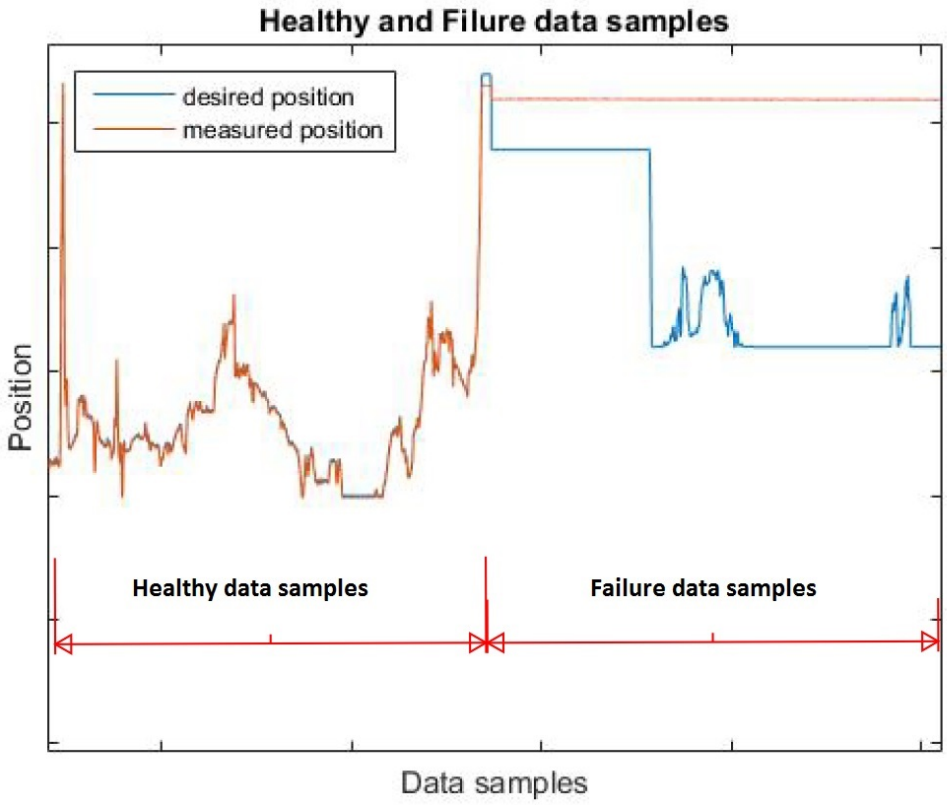


Figure 1.4 *Illustration of healthy and failure data. Axis values are anonymized.*

them to classify the FIHDS's out of healthy data. For this reason the first part is crucial since firstly the labels are needed for supervised learning machine to learn and secondly if the wrong labels are taught to supervised learning machine it will perform wrong by default.

1.3 State of the art

When options for predicting the upcoming failure of physical asset were examined then following concepts came across, seeking more and less the same procedure as pursued here: Intelligent maintenance support system (IMSS) [19], Early Warning System (EWS) [15], Decision Support Systems (DSS) [19], Predictive data analysis, Failure forecasting, On-line failure prediction [17].

All the concepts above are more or less trying to do the same, that is predicting the failure beforehand. Some of the concepts take a stand about the future usage of the prediction like DDS and IMSS. In this study the focus was not on the further

use of the prediction but about the feasibility of machine learning methods for the failure prediction of physical asset and specially of aeroplane flight control surface. When the methods to solve the problem were examined the following concepts came across:

- Artificial Neural Network (ANN), Logistic Discrimination (LD), Decision tree (DT), Bayesian probability network, Support Vector Machine (SVM) and Neuro-fuzzy model (NF) [15]: In this paper the artificial neural network was used to attempt to build early warning system for predicting signal for possible economic crisis.

- Back Propagation Neural Network, Self-Organizing Map (SOM) and Principal component analysis [13]: In this paper SOM and Back Propagation NN were used in order to predict the rolling element bearing remaining useful life.

- Recurrent neural network, Analytic hierarchy process and Petri Nets [19]: In this paper the intelligent predictive decision support system for a power plant was build for a support for condition based maintenance by using the above machine learning methods.

- Jordan Network [14]: In this paper the neural network capability in general level to predict failures was studied.

- Self Organizing maps (SOM) and Principal component analysis [8]: In paper the data measurement methodologies and the usability of SOM for failure prediction for aeroplane engine failures was studied.

All of the methods above are not generally classified as a Machine Learning methods, but some are rather called as conventional data analysis methods. They are listed here because they have been used to solve similar problem that is the research question here. In this paper it was chosen to use specifically Machine Learning methods for case study failure forecasting. The exception will be K-mean clustering which belongs in practice more in group of conventional data analysis methods rather than in Machine Learning methods. Nevertheless K-mean used among with Machine Learning methods can produce more value to the analysis, which we will find out later.

1.4 Input selection

Input selection is a critical step when adapting a machine learning methods. For example Selfner M et. al.[17] claimed that "...issue of choosing a good subset of input variables has a much greater influence on prediction accuracy than the choice of modeling technology." Input space of the learning machine should be as compact as possible since all unnecessary inputs will generate a modelling error [10]. On the other hand all parameters related to the issue should be included in order to have best possible estimate.

In this study from several flight process data parameters only the ones directly related to the flight control surface operation was selected. Some parameters were not mutually dependent and thus the input parameter space was further reduced by combining dependent parameters analytically. The whole input parameter selection and reduction process was done with coo-operation of domain experts in order to achieve maximum informative and on the other hand minimal confusing input parameter space for learning machines.

2. IDENTIFYING THE FAILURE INDICATING HEALTHY DATA SAMPLES (FIHDS) FROM THE HISTORICAL DATA OF HEALTHY SYSTEM

2.1 Methods

Self-Organizing Maps (SOM)

Self-Organizing Map (SOM) is a one type of unsupervised learning machine [12]. In practice SOM is a dimensionality reduction method. It is an algorithm which have N dimensional feature input and returns M dimensional feature output, where $N > M$ and in many practical case $N >> M$. N dimensional feature input means simply the dataset which has the quantity of N parameters (see figure 1.2). M dimensional feature output is on the other hand a dataset with M parameters which usually does not correspond with any physically consistent properties.

Regardless the fact that SOM transforms data with number of physically consistent parameter into some smaller number of physically non-consistent parameters, still SOM have some advantageous properties. In order to illustrate the statement lets see the figure 2.1. Lets assume that the data of figure 2.1 have been measured out of physical system and all of the 10 data parameters are physically consistent. Since the data is in 10-dimensional is hard to find any patterns in the data just by looking at it. In order to see some patterns in the data it need to be reshaped it into more illustrative form and SOM provides a tool for this.

Now lets perform a dimensionality reduction with SOM from $N = 10$ dimensionality to $M = 2$ dimensionality. The reason for this is simple. Humans can easily visualize patterns with dimensionality equal 3 or less. On the other hand dimension 2 is

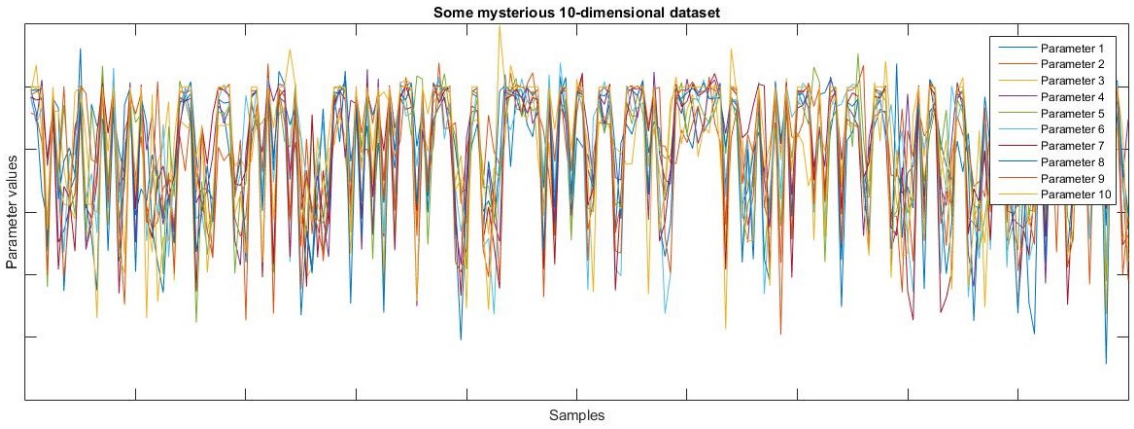


Figure 2.1 *Illustration of some artificial 10-dimensional example dataset.*

suitable for presentations on paper or on computer screen. The result is presented in figure 2.2.

Figure 2.2 illustrates the size of 10 x 10 neuron SOM neural network and the colors of each neurons are illustration of the final weight value distance to the neighbour weight of each neurons in the network. The two dimensions of the graph (vertical and horizontal) are physically non-consistent and thus meaningless to us. Still it can be clearly seen some information in this 2D SOM of figure 2.2.

The observation based on the figure 2.2 is that the data might be twofold. Now if we were told that the original data have been measured from the pine-woods of Siberian then we might think that there might be actually two types of pine-woods instead of one involved in the study. If we are told that the data have been the monitored from some sawmill then we may conclude that the sawmill may run in two different modes for some reason or another.

When we trained the SOM of 2.2 we also recorded the Best Matching Units (BMU's) for each samples (see section 2.1). Thus every sample of figure 2.1 is connected to one neuron of network in of figure 2.2. Now if the original data was from pine-woods we are able to separate the original samples into two classes based on BMU's. The same would apply on the sawmill and where we would be able to separate the data samples of two modes.

The research question of this Part 1 mimic exactly the example problem described above. Here we have an enormous dataset containing millions data samples with

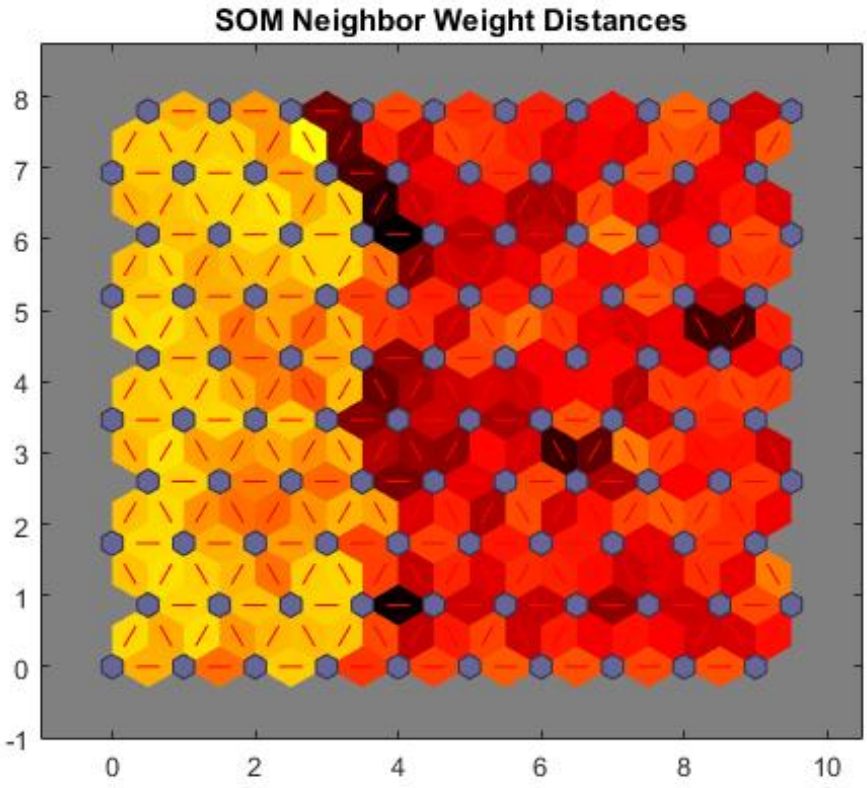


Figure 2.2 *2D SOM illustration of the data from figure 2.1.*

tens of parameters. We assume that the healthy data is composed from FIHDS's and PHDS's but we do not exactly know which samples are which ones.

SOM map and the final weights of it will be generated during the training by introducing data samples to SOM. Each sample generates the highest output to one neuron of the SOM. That neuron will become the Best Matching Unit (BMU) and will have the greatest weight update. Also the neighbours of that particular neuron will be updated in such way that the neurons close to it will have a great update and the neurons far away from it will have a small update. Thus each sample introduced to SOM "drags" weights of the whole network towards its BMU (see fig 2.3). After the procedure have been done with all samples the result be the final trained SOM. The final distances of the weights of the trained SOM can be calculated and used for illustrative manner as demonstrated in figures 2.2 and 2.5

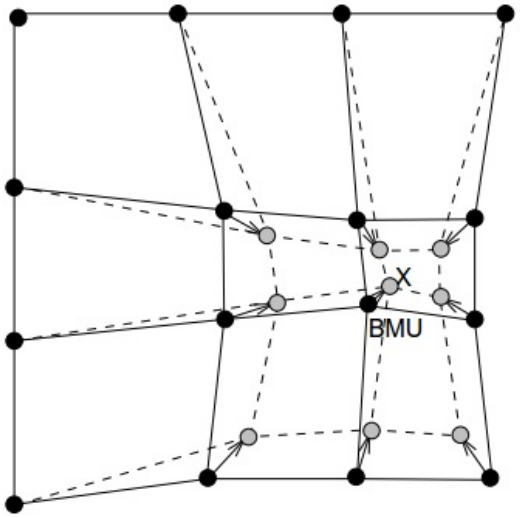


Figure 2.3 *The intuitive illustration of SOM weight update caused by sample x. Solid lines are presenting situation before the update and dashed lines are presenting the situation after the update [18].*

Theory of SOM

Describing SOM Lets consider some black box for doing some dimensionality transform from N dimensional input space to M dimensional output space, where $N, M \in \mathbb{N}^+$. Now if the black box is SOM then the following conditions will be satisfied:

1. $N > M$ and usually $M = 2$.

2. There exists some pre defined number K of *neurons*, where $K \in \mathbb{N}^+$.

3. Each neuron has a one or more neighbours in M dimensional space, where distances between the neurons are measurable.

4. For every neuron there exists one connection from every dimension of input space (see figure 2.4).

5. The connections are weighted with weights $w_n^{(k)}$, where $w \in R$ and $n = 1...N$ and $k = 1...K$. Weights $w_n^{(k)}$ can be also seen as an N x M matrix W.

Training SOM ,

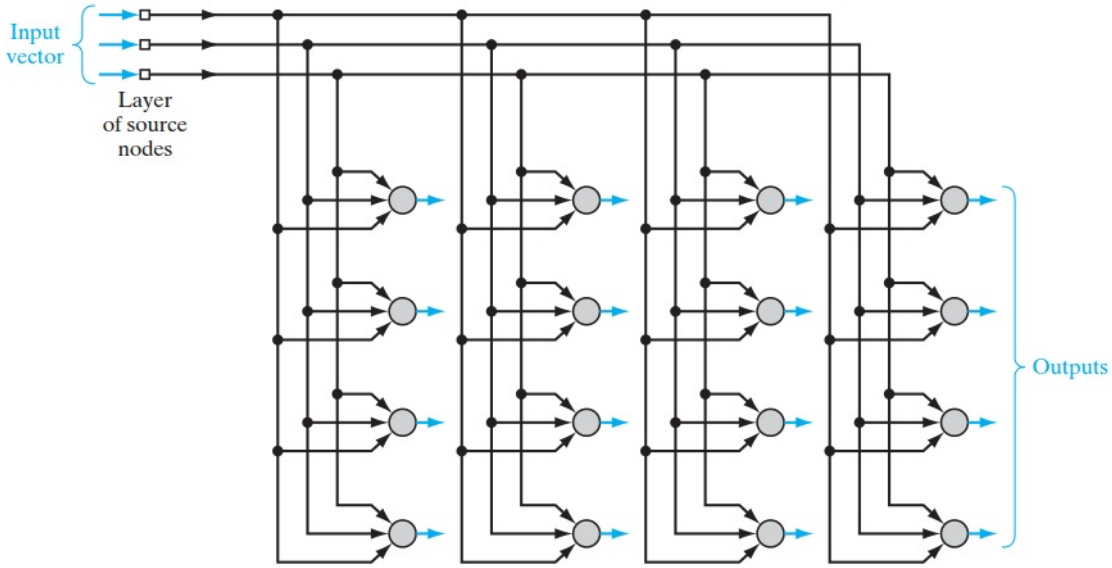


Figure 2.4 *Illustration of two dimensional SOM connected to three dimensional input space [12]*

1. Initialize a set of random weights $w_n^{(k)}$. A good choice is close to zero but non-zero.

2. Have a randomly picked sample vector $x = [x_1, x_2, ...x_N]$ from your input data space.

3. Find the Best Matching Unit neuron (BMU), which has the closest distance between the input vector x and weight vector $w^{(k)}$ in such way that

$$\| x - w^{(BMU)} \| = \min_{k=1...K} \{\|x - w^{(k)}\|\} \tag{2.1}$$

where $\| \cdot \|$ is the euclidean distance measure defined as

$$\|x - w^{(k)}\| = \sqrt{\sum_{i=1}^{N} (x_i - w_i^{(k)})} \tag{2.2}$$

4-A. Update the weights of SOM iteratively such as the new weights become as

$$w^{(k)}(t+1) = w^{(k)}(t) + \alpha(t) h_{BMU,k}(t)[x(t) - w^{(k)}(t)] \tag{2.3}$$

where t is a time (in practise a step of iteration), $\alpha(t)$ is a *learning rate function* and $h_{BMU,k}(t)$ is and *neighbourhood function.*

Repeat 2, 3 and 4 iteratively by each time excluding the sample x.

4-B Or update the weights using batch training algorithm

$$w^{(k)}(t+1) = \frac{\sum_{i=1}^{D} h_{BMU,k}(t)x_i}{\sum_{i=1}^{D} h_{BMU,k}(t)} \tag{2.4}$$

where D is the number of data samples.

Some options for *neighbourhood functions* are:

- Dubble: $h_{BMU,k}(t) = \mathbf{1}(\sigma_t - d_{BMU,k})$
- Gaussian: $h_{BMU,k}(t) = e^{-d^2_{BMU,k}/2\sigma_t^2}$
- Cutted Gaussian: $h_{BMU,k}(t) = e^{-d^2_{BMU,k}/2\sigma_t^2}\mathbf{1}(\sigma_t - d_{BMU,k})$
- EP: $h_{BMU,k}(t) = max\{0, 1 - \mathbf{1}(\sigma_t - d_{BMU,k})^2\}$

where σ_t is the neighbourhood radius at time t, $h_{BMU,k}(t) = \|r_{BMU} - r_k\|$ is the distance between BMU and neuron k, and $\mathbf{1}(x)$ is the step function [18].

Some options for *learning rate functions* are:

- Linear: $\alpha(t) = \alpha_0(1 - t/T))$
- Power: $\alpha(t) = \alpha_0(0.005/\alpha_0)^{t/T}$
- Inverse: $\alpha(t) = \alpha_0/(1 - 100t/T))$

where T is the training length and α_0 is the initial learning rate [18].

K-mean clustering

K-mean clustering is one of the most primitive way to cluster data and thus it is also one of the most simplest unsupervised learning methods. In this method the

only objective is to minimize $\sum_{k=1}^{K} \sum_{x_n \in S_k} \|x_n - \mu_k\|^2$ with respect to μ_k, where x_n are the data samples, μ_k are the cluster centres and S_k are sets of clusters. The minimization can be done relatively straightforward by Lloyd's algorithm [6].

In this algorithm there is an initialization step and two iteration steps described next.

Initialization step: Have a choice a number K of cluster centres labelled as μ_k and place them in to the same space as your original data. A usual approach to initialize the positions of the clusters is to pick amount of K random data samples and have the location of them to presenting the locations of cluster centres.

Step 1: Calculate the distances between all cluster-centres and data samples and then label each data sample x_n to belonging in the cluster center closest to it.

$$S_k = \{x_n :\| x_n - \mu_k \| \leq \| x_n - \mu_l \|, \forall l \in K/k\} \tag{2.5}$$

Step 2: Move the cluster centres μ_k to the center of mass of the data samples labelled to the cluster.

$$\mu_k = \frac{1}{|S_k|} \sum_{x_n \in S_k} x_n \tag{2.6}$$

Repeat Step 1 and Step 2 until any data sample does not change its cluster membership, or stop earlier. The algorithm converges to local minimum [6] thus several runs with different cluster center initializations may be in order to find the best solution.

2.2 Results

Self-Organizing Maps (SOM)

The SOM calculations here were mostly carried out by using the functions of *SOM Toolbox for Matlab 5* [5]. Here all healthy data of Aeroplane A before the first failure have been used to train the SOM. This includes all 40 healthy flights plus the healthy part of the first failure flight. Thus there was no information about the

failure available for SOM. The aim here was to see if the SOM would cluster the data-points near to the upcoming failure separate from the rest of the data. If so then:

1. We would have a proof that FIHDS's exists.

2. We would have some notion about which part of the data would be FIHDS's

3. We would know than we can at some level classify FIHDS's out of rest of the healthy data by machine learning methods.

'Symptom 5901': Several SOM configurations with different parameters were configured and trained in order to detect FIHDS's. This try and error approach led to the beneficial SOM configuration described in table 2.1.

Table 2.1 *Description of the SOM capable to reveal FIHDS's*

SOM dimensionality (output space / M)	2
Number of neurons	6018
Training algorithm	Batch
Neighbourhood function	Gaussian
Topological neighbourhood	Hexagonal

The U-matrix of trained SOM is presented in figure 2.5.

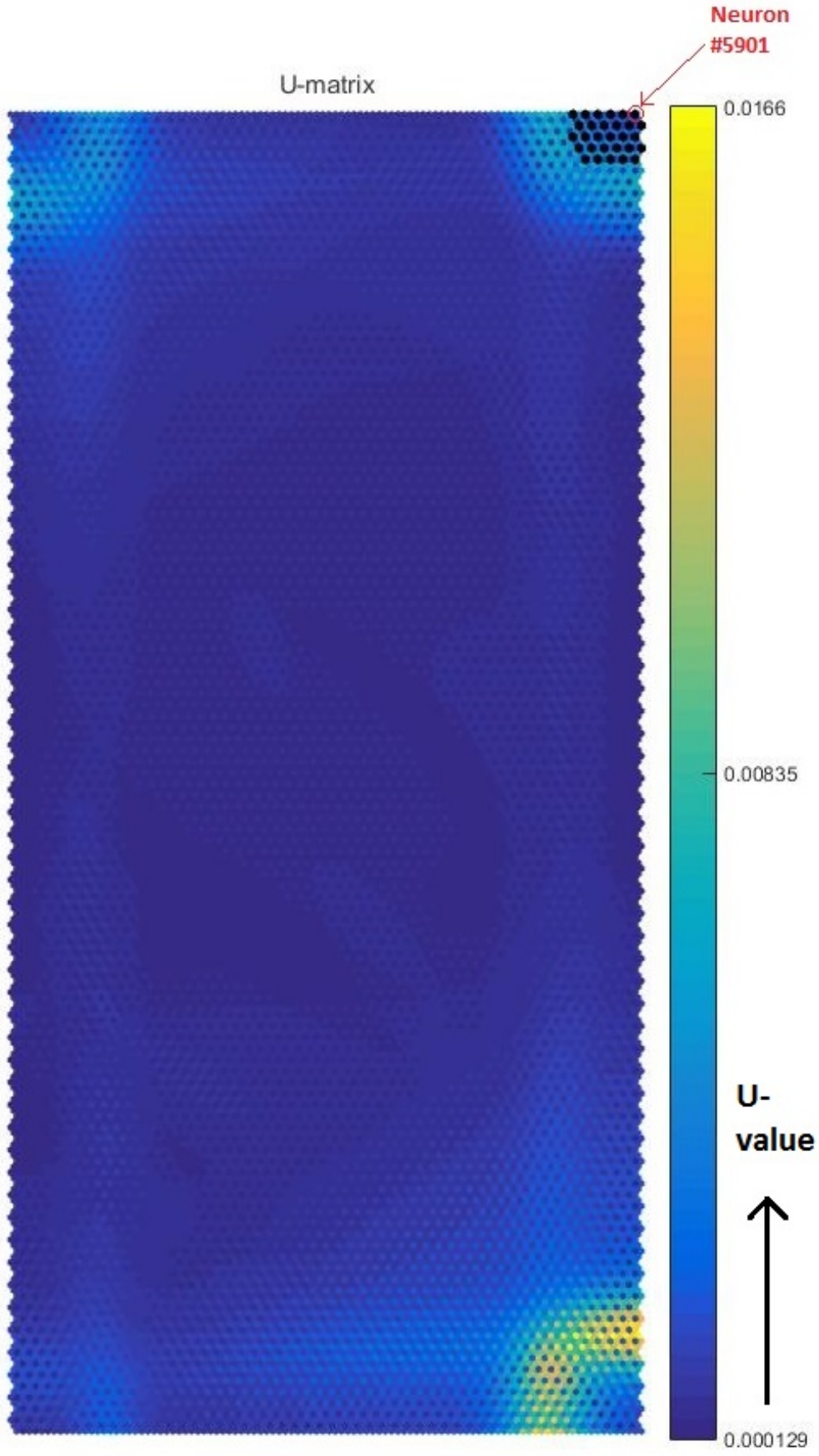


Figure 2.5 *U-matrix of the SOM capable to reveal FIHDS's*

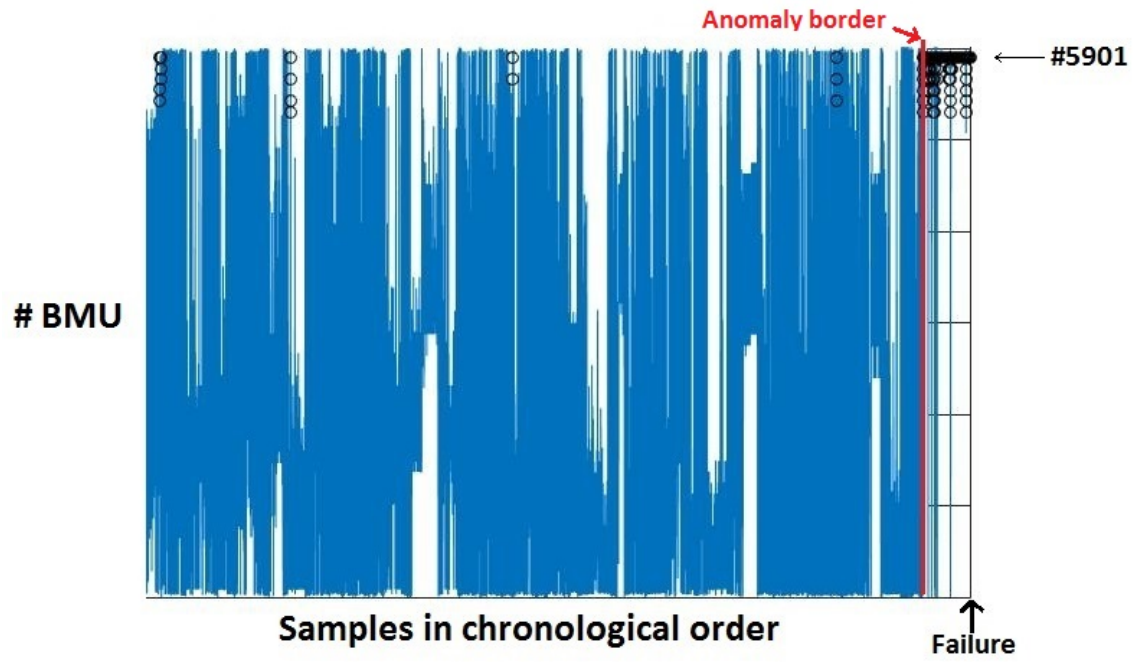


Figure 2.6 *Illustration of BMU's for final parts of the data samples of the Aeroplane A presented in chronological order.*

The U-value that is also the color scale of figure 2.5 presents the final weight distances between the neighbouring map units of trained SOM. Thus for example the blue neurons in the lower right corner are relatively far away from the blue neurons in the centre of the map since there stands a region of yellow neurons between them having great distances to their neighbours.

Here the absolute values of distances u are not the interest, but instead the fact that there exists three clusters in three different corners of this SOM. The clusters of the corners are relatively far away from the rest of the neurons since $u = 0.0166$ is 10^2 scale bigger than $u = 0.000126$. The meaning of clusters in upper left corner and lower right corner remained here unknown. Instead the meaning of the cluster of the upper right corner can be rationalized by observing the figure 2.6. The black dots in figure 2.5 are presenting the same neurons as the black circles in figure 2.6.

In figure 2.6 the BMU's are distributed uniformly among the data samples until the anomaly border. After the anomaly border the samples will mostly have as a BMU the neurons that are clustered in upper right corner of the SOM of figure 2.5.

Special attention need to be paid to neuron # 5901, since after the anomaly border

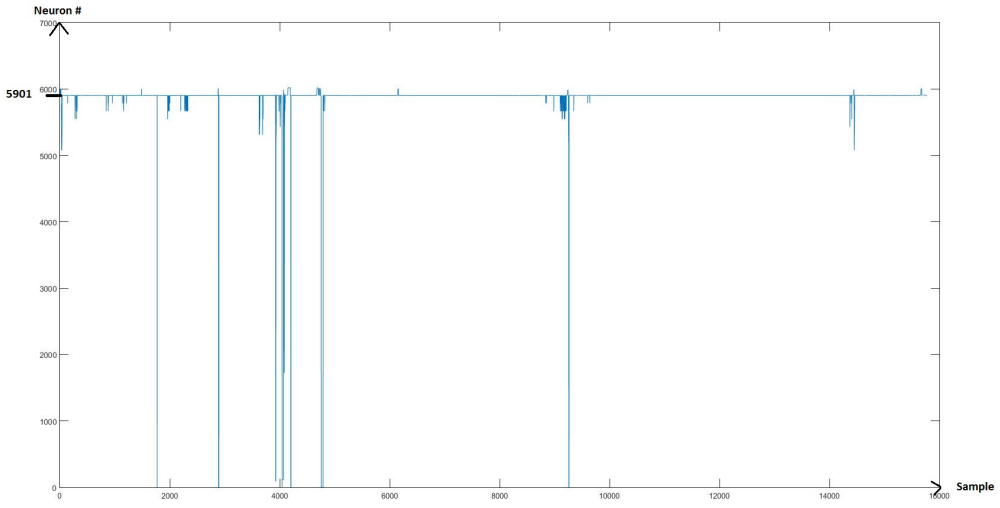


Figure 2.7 *BMU's for data samples after the anomaly border of figure 2.6.*

in figure 2.6 the neuron # 5901 became as a BMU for 88 % of the data samples. For comparison for data samples before anomaly border the neuron # 5901 becomes as a BMU only 2 % of the cases.

In the data before the anomaly border the neuron # 5901 becomes as a BMU maximum 1.2 s in a row. Instead after the anomaly border the neuron # 5901 becomes as a BMU average 5.5 s in row and hawing a maximum duration of 7.3 min in row. The BMU's after the anomaly border of figure 2.6 have been presented in figure 2.7.

The distribution of BMU's after the anomaly border compared to the time before the anomaly border can be also seen from the histogram of figure 2.8.

K-mean clustering

K-mean clustering was used to examine the healthy historical data of Aeroplane A. The aim was to find a cluster(s) which might present FIHDS's. After trying clustering with a several number of clusters it was found that cluster quantity of three was the most illustrative. The result is presented in figure 2.9.

The blue curvature in figure 2.9 is just for illustration and presents the position of one control surface. The brown curvature describes in which cluster each data

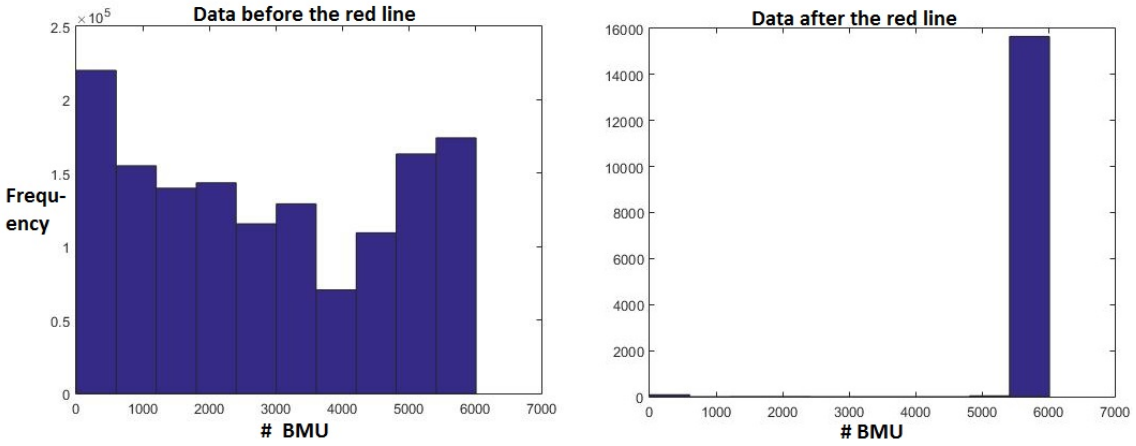


Figure 2.8 *The histogram presenting the BMU occurrence for the flight data of Aeroplane A, before the anomaly border of figure 2.6 (left) and after the anomaly border (right).*

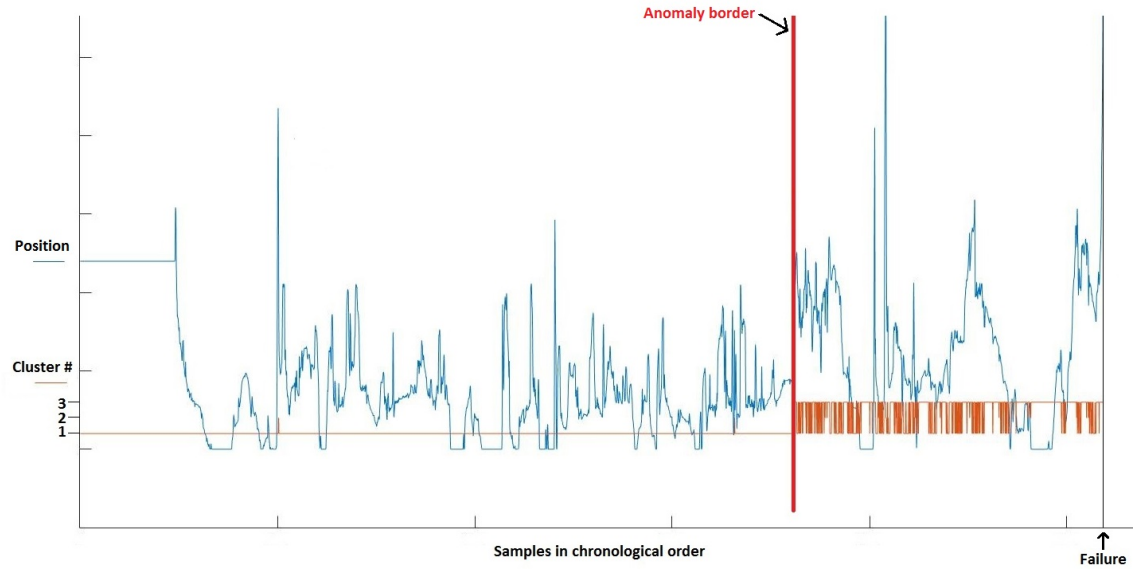


Figure 2.9 *Data classification in three cluster by K-mean clustering.*

sample belongs. The anomaly border in figure 2.9 lies on same real time moment as the anomaly border in figure 2.6. Before the anomaly border there was no data samples clustered in cluster no. 3 and after the anomaly border there were approximately half of the samples clustered in cluster no. 3.

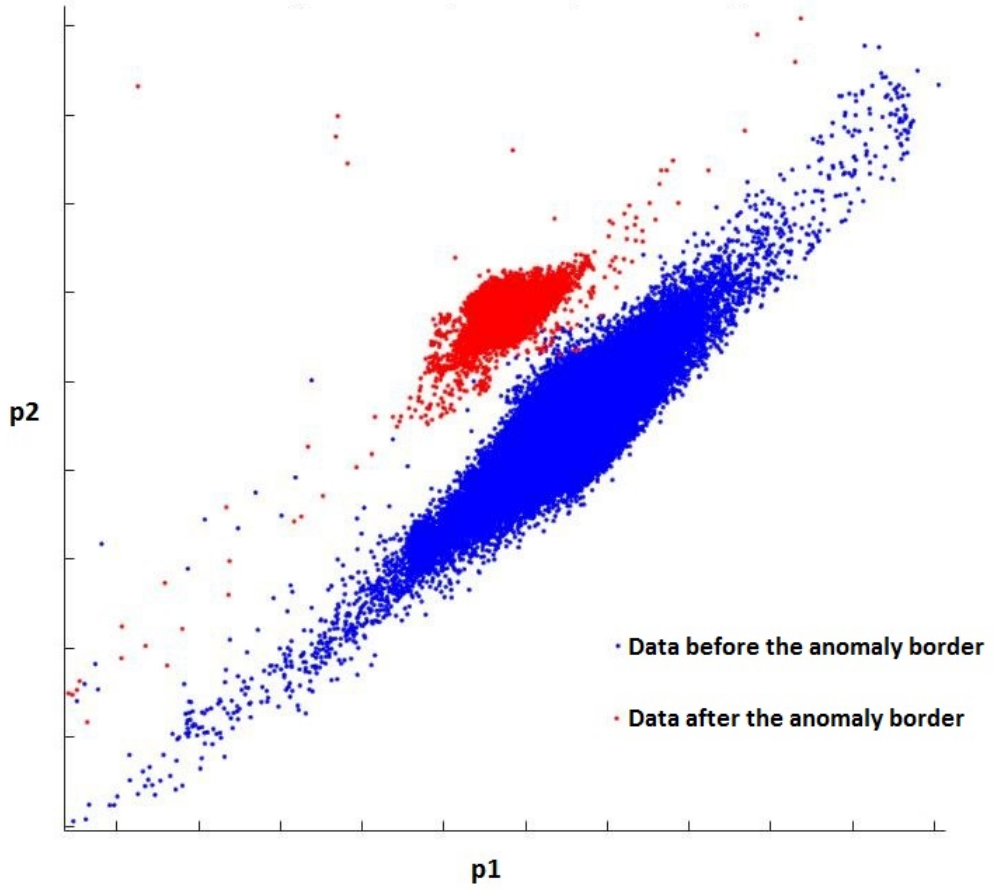


Figure 2.10 *Aeroplane A flight data before and after a anomaly border.*

Heuristic approach

The objective here was to construct a simple graphical illustration in order to be confirmed that the data before the anomaly border and the data after anomaly border indeed are somehow different. Here several parameters of the original data were combined mathematically into the a parameters p1 and p2 in intuitive way. The parameter pairs (p1,p2) have been plotted into 2D graph presented in the figure 2.10. Blue dots in the figure are generated from the data samples before the anomaly border of figures 2.6 and 2.9 and red dots are generated from the data after the anomaly border.

2.3 Conclusions

This far some anomalies in the flight data before the failure have been found. Those anomalies are the data samples having neuron #5901 frequently as a BMU in SOM (see figure 2.6) and the data samples clustered no.3 in K-mean clustering (see figure 2.9. Since these anomalies exists before the failure but after the long period of normal operation it is reasonable to assume that those data samples are carrying some information about the upcoming failure and thus indeed are FIHDS's.

There exists also some data samples between the anomaly border and failure from which any anomalies were not detected by the methods of this study. Now when considering the aim of the Part 2 of this study that is to build a learning machine which can predict the future failures from the new data, it would be preferable to have a machine which is able to classify any data just before the failure in separate class from the pure healthy data not just the data that the specific SOM and K-mean clustering configuration used here did saw abnormal. In order to achieve this the assumption was mate that all the data between the anomaly border and failure are FIHDS's.

The intuition why all the data between the anomaly border and failure should be classified as a FIHDS's when teaching them to a learning machine may be as follows: You ride a bicycle. If the bicycle is running smooth and nice you consider it working. Then suddenly the front bearing of your bicycle starts to make weird noise and you consider it to be failing soon. Then the weird noise stops, then it starts, then it stops, then it starts... Even the weird noise temporally stops you still might consider all the time that your bicycle will fail soon. Now if we are monitoring our bicycle with learning machine we would like that the machine will see the data sequences between the weird noise sequences also as a FIHDS's if the bicycle is really failing.

The results of heuristic approach presented in figure 2.10 confirms that there indeed exists two separate classes of healthy data.

Now the healthy data of Aeroplane A can be seen in two separate classes. Based on this result a learning machine which may be able to classify the new data in FIHDS's and PHDS's and in this way giving the indication about the upcoming failure can be build.

From the actual history of Aeroplane A it is known that the anomaly border of

figures 2.6 and 2.9 stands at the moment when the Aeroplane A failure flight I started. In practice the time interval between the anomaly border and failure was several tens of minutes.

3. TEACHING SUPERVISED LEARNING MACHINE TO DETECT FIHDS'S OUT OF NEW DATA

The first part of the study (chapter 2) resulted something that can be now on to be taught to the supervised learning machine. Those are the labels PHDS's and FIHDS's and now on the supervised learning machine can be build to make distinction between PHDS's and FIHDS's from the new data.

By observing previous chapter in can be noted that the learning machines already exist here which can make the distinction between the PHDS's and FIHDS's. Those are SOM, K-mean clustering and Heuristic approach which were just used to find FIHDS's.

In case of SOM the ready trained weight matrix W of SOM can make the distinction between the PHDS's and FIHDS's and in future new data sample could be plugged in and see where the BMU's will land. If they land on the upper right corner of the SOM or specially on the neuron # 5901 the conclusion might be drawn that the sample is FIHDS. And if this happens in frequency exceeding some predefined threshold the indication about the upcoming failure could be concluded.

For example in case of K-mean clustering there is an option to examine in which cluster the new sample belongs. If it belongs to the cluster no. 3 then it may be concluded that the sample must be a FIHDS. And once again if this happens in frequency exceeding some predefined threshold there exists some indication about the upcoming failure.

For example in case of heuristic approach the new sample may be plotted in the graph presented in figure 2.10. If the sample land on among the group of FIHDS it may be concluded that the new sample must be also a FIHDS. And if this happens in frequency exceeding some predefined threshold the indication about the upcoming

failure may be interpreted.

This approach is correct but there exists a problem with it, and the core aim of this Part 2 is to overcome this problem. In order to see the problem the main objective of this study should be considered, that is trying to predict the future and specially predict the upcoming failure. Thus machine which in **general** can separate FIHDS's out of healthy data is needed.

In the first part (Chapter 2) FIHDS label was not only given for the anomaly samples found with the methods used there but also for the data samples between the found anomaly samples. Now when this extended set of FIHDS's will be taught to the supervised learning machine the machine will predict failures in more general level.

3.1 Methods

3.1.1 Supervised learning machines

Radial Basis Function (RBF)

Radial Basis Function (RBF) is a non linear classifier. RBF function has a basis function which measures the radial distance of the *observable data sample* x to the e*xample data sample* y. In learning machine approach the distances of samples are summed in order to find out the similarity between the observable data set X and the example data set Y. Based on the similarity of the data sets and the previous knowledge about the example data set Y the conclusions about the observable data set X can be mate. In supervised learning the example data set Y may be a set of historical data or a smaller set of clusters generated from the historical data.

The actual radial basis function is defined as

$$F(x) = \sum_{n=1}^{N} w_n \varphi(\|x - x_n\|) \tag{3.1}$$

and for classification

$$F(x) = sign\left(\sum_{n=1}^{N} w_n \varphi(\|x - x_n\|)\right) \tag{3.2}$$

where $N \in \mathbb{N}^+$ is the number of samples, w_n's are weights and the $\|x - x_n\|$ are radius's between some point x and data sample x_n [12]. The *basis function* φ has a several forms [12] like:

1. Multiquatratic: $\varphi(\|x - x_n\|) = \sqrt{\|x - x_n\|^2 + c^2}$, where $c > 0$.

2. Inverse multiquatratic: $\varphi(\|x - x_n\|) = \frac{1}{\sqrt{\|x-x_n\|^2+c^2}}$, where $c > 0$.

3. Gaussian: $\varphi(\|x - x_n\|) = exp(-\lambda\|x - x_n\|^2)$, where $\lambda > 0$.

Here the goal is to have a $F(x)$ which describes the system behaviour in hands. In this case it must satisfy the equality $F(x_i) = y_i$. In other words for some data sample x_i the function $F(x_i)$ produces the output y_i which correspond the actual output of the system. Thus for some data sample i the equality

$$\sum_{n=1}^{N} w_n \varphi(\|x_i - x_n\|) = y_i \tag{3.3}$$

applies. Now if the i goes through all the data samples $i = 1...N$ then a set of equations are generated by the equation 3.3. This set of equations can be expressed in the matrix form as:

$$\begin{bmatrix} \varphi(\|x_1 - x_1\|) & \varphi(\|x_1 - x_2\|) & \dots & \varphi(\|x_1 - x_N\|) \\ \varphi(\|x_2 - x_1\|) & \varphi(\|x_2 - x_2\|) & \dots & \varphi(\|x_2 - x_N\|) \\ \vdots & \vdots & \vdots & \vdots \\ \varphi(\|x_N - x_1\|) & \varphi(\|x_N - x_2\|) & \dots & \varphi(\|x_N - x_N\|) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{3.4}$$

Equation 3.4 can be rewritten simply as $\phi\vec{w} = \vec{y}$, and from this equation the weights w_n can be solved as:

$$\vec{w} = \phi^{-1}\vec{y} \tag{3.5}$$

RBF with K-mean clustering: In practice solving the equation 3.5 may be computationally demanding with large datasets, since inverting large matrix is computationally heavy task. Pre size of the set of solvable equations in 3.4 may be reduced by having equations 3.1 an 3.2 in form of

$$F(x) = \sum_{n=1}^{K} w_k \varphi(\|x - \mu_k\|) \tag{3.6}$$

and for binary classification

$$F(x) = sign\left(\sum_{n=1}^{K} w_k \varphi(\|x - \mu_k\|)\right) \tag{3.7}$$

where $K \in \mathbb{N}^+$ and $K << N$.

The points of μ_k can be chosen by many ways but practice have been shown that by choosing the centres of K-mean clustering (see sec. 2.1) as a μ_k's the equation 3.3 approximately holds

$$\sum_{n=1}^{K} w_k \varphi(\|x_i - \mu_k\|) \approx y_i \tag{3.8}$$

Thus when i goes through the whole dataset $i = 1...N$ then equation 3.8 generates a set of equations which can be expressed in matrix form as:

$$\begin{bmatrix} \varphi(\|x_1 - x_1\|) & \varphi(\|x_1 - x_2\|) & \dots & \varphi(\|x_1 - x_K\|) \\ \varphi(\|x_2 - x_1\|) & \varphi(\|x_2 - x_2\|) & \dots & \varphi(\|x_2 - x_K\|) \\ \vdots & \vdots & \vdots & \vdots \\ \varphi(\|x_N - x_1\|) & \varphi(\|x_N - x_2\|) & \dots & \varphi(\|x_N - x_K\|) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} \tag{3.9}$$

or simply

$$\phi\vec{\omega} = \vec{y} \tag{3.10}$$

Since $K < N$ the ϕ matrix of equation 3.10 is not a square matrix and thus it is not invertible. Still feasible solution can be achieved by pseudo-inverse and thus

$$\vec{w} = pinv(\phi)\vec{y} \tag{3.11}$$

Now if the weights $\vec{w}$ from the equation 3.11 will be solved then a label y_{new} can be calculated for new data sample x_{new} by using equation 3.8 $y_{new} \approx \sum_{n=1}^{K} w_k \varphi(\|x_{new} - \mu_k\|)$.

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a binary classification algorithm capable to classify a set of data in to two classes [12]. The idea is to find a hyperplane which separates a data into a two classes in some hyperspace. SVM differs from other bi-

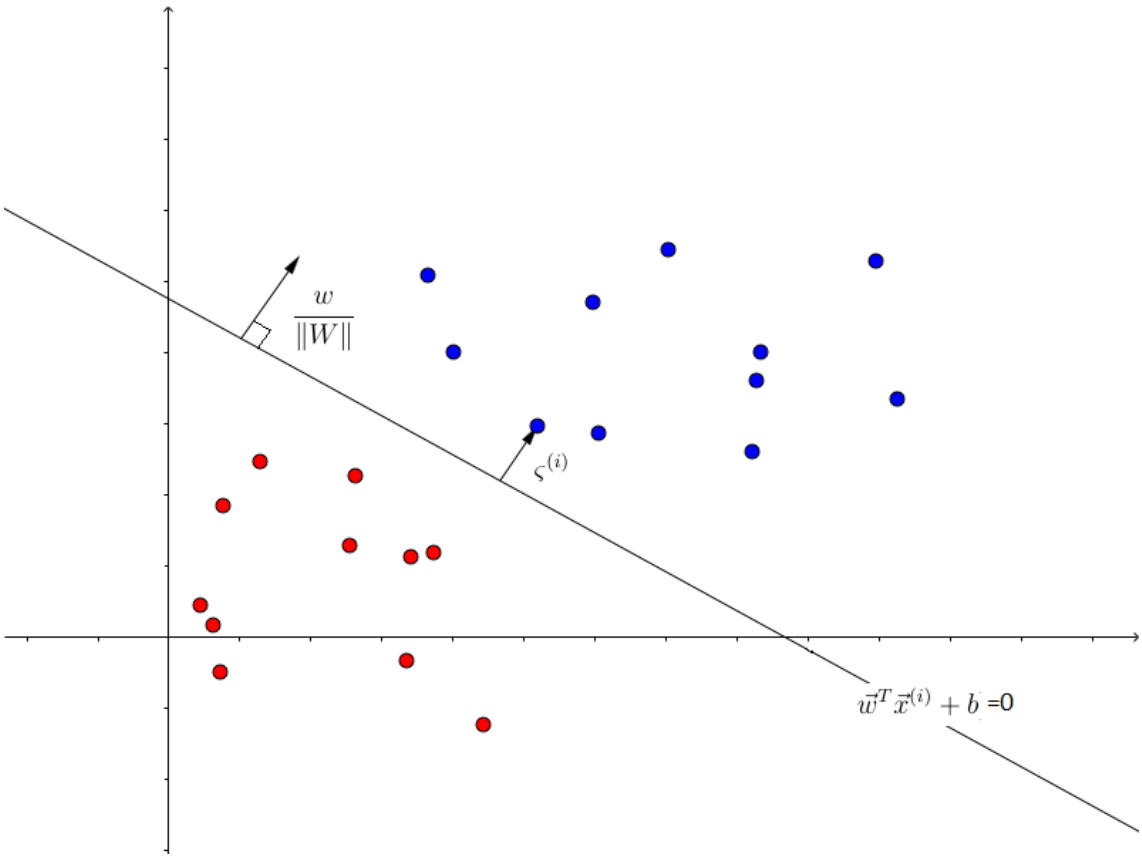


Figure 3.1 *Illustration of 1-D 'hyperplane' in 2-D 'hyperspace' separating the some linearly separable data in two classes by having the margin ζ.*

nary classifiers in the way that it finds the hyperplane having the maximum margin between the two classes. The data samples (vectors) which are touching the margin are called as a *support vectors*.

Lets donate data sample as $\vec{x}$ where $\vec{x} \in R^n$ and n is the dimensionality of the feature space (i.e number of parameters of data, see fig 1.2) of the input data vector.

Let first assume that the data set is linearly separable in two classes and thus the separation can be done by the hyperplane written as follows:

$$\vec{w}^T\vec{x} + b = 0 \tag{3.12}$$

Now let i denote the i:th *sample* of the data (see fig 1.2) and let denote the desired response as $y_i \in \{-1, +1\}$ for each data sample $x^{(i)}$.

Thus it may be written:

$$\begin{cases} \vec{w}^T\vec{x}^{(i)} + b \geq 0 & \text{when } y_i = 1, \\ \vec{w}^T\vec{x}^{(i)} + b < 0 & \text{when } y_i = -1, \end{cases} \tag{3.13}$$

The data can be scaled here without having none of the samples changing its label. Thus the margin between the two separate classes can be freely scaled and the margin can be required to be 1. Now following equation is consistent with the equation 3.13.

$$\begin{cases} \vec{w}^T\vec{x}^{(i)} + b \geq 1 & \text{when } y_i = 1, \\ \vec{w}^T\vec{x}^{(i)} + b < 1 & \text{when } y_i = -1, \end{cases} \tag{3.14}$$

On the other hand the equation 3.14 is consistent with the equation

$$y_i(\vec{w}^T\vec{x}^{(i)} + b) \geq 1 \tag{3.15}$$

In other words the classification have been done correctly when the equation 3.15 holds.

The vectors that are on the margin are called the support vectors and for those vectors it applies that $y_i(\vec{w}^T\vec{x}^{(i)} + b) = 1$ and thus

$$y_i(\vec{w}^T\vec{x}^{(i)} + b) - 1 = 0 \tag{3.16}$$

The width of the decision boundary is

$$\varsigma = |x_s^{(-1)} - x_s^{(+1)}|\frac{\vec{w}}{\|\vec{w}\|} \tag{3.17}$$

where $x_s^{(-1)}$ is some support vector from -1 side of the decision boundary, $x_s^{(+1)}$ is some support vector from $+1$ side of the boundary and $\frac{\vec{w}}{\|\vec{w}\|}$ is the normal unit vector for the boundary. Now by substituting equation 3.16 into equation 3.17 will produce the result

$$\varsigma(\vec{w}) = \frac{2}{\|\vec{w}\|} \tag{3.18}$$

Since the SVM is maximum margin binary classifier the objective here is to maximize the width ς of the equation 3.18. For mathematical convenience the minimization should be rather performed on

$$\varsigma(\vec{w}) = \frac{1}{2}\|\vec{w}\|^2 \tag{3.19}$$

The equation 3.19 is consistent with the equation 3.18 since

$$\max_{\vec{w},b} \frac{2}{\|\vec{w}\|} \Leftrightarrow \min_{\vec{w},b} \|\vec{w}\| \Leftrightarrow \min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|^2 \tag{3.20}$$

Now the objective is to find extremum of the function 3.19 by having the constrain 3.16. For this purpose the Lagrange Multiplier [11] suits well [12]. For equation 3.19 with constraint 3.16 the Lagrangian function L will be

$$L(\vec{w}, b, \alpha) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left[(y_i(\vec{w}^T\vec{x}^{(i)} + b) - 1\right]) \tag{3.21}$$

where m is the number of the samples and $\alpha_i \geq 0$:s are some Lagrangian multipliers. The extremum of the Lagrangian can be solved by finding the minimum with respect to $\vec{w}$ and b and then maximum with respect to α. In order to find the extremum there exists two conditions of optimality:

$$\frac{\partial L(\vec{w}, b, \alpha)}{\partial \vec{w}} = \vec{0} \tag{3.22}$$

and

$$\frac{\partial L(\vec{w}, b, \alpha)}{\partial b} = \vec{0} \tag{3.23}$$

From condition 3.22 it can be derived that

$$\vec{w} = \sum_{i=1}^{m} \alpha_i y_i \vec{x}^{(i)} \tag{3.24}$$

and from condition 3.23 it can be derived that

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{3.25}$$

Substituting equations 3.24 and 3.25 back in to the equation 3.21 will produce:

$$L(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \vec{x}^{(i)T} \vec{x}^{(j)} \tag{3.26}$$

Finding the minimum of the equation 3.26 with respect to α is the dual problem for the finding the maximum with respect to $\vec{w}$ and b and the further finding minimum with respect to the α of the equations 3.21 [7].

Now if some optimum α_o have been solved from the equation 3.26 then the optimal $\vec{w_o}$ may be solved from the equation 3.24.

The optimal solution is the problem of *optimization*. One efficient way to perform the optimization here is quadratic programming. Quadratic programming is not discussed here since it is out of the scope of this study.

SVM with soft margin: For data not linearly separable or noisy a soft margin is needed. Soft margin allows to some samples to violate the margins of the decision boundary. The soft margin can be added in SVM by using a error therm $\epsilon \geq 0$. When the error therm is added in equation 3.19 then new subject of minimization will be become as

$$\varsigma(\vec{w}) = \frac{1}{2} \|\vec{w}\|^2 + C\vec{\epsilon} \tag{3.27}$$

where $C \geq 0$ is some scaling constant referred here as a *Box Constraint*. When an error term is added in constraint 3.16 then new boundary will be become as

$$y_i(\vec{w}^T \vec{x}^{(i)} + b) - 1 + \vec{\epsilon} = 0 \tag{3.28}$$

This subject of minimization and boundary will generate exactly the same results from Lagrangian functions as above except the Lagrangian multipliers are limited in such way that $0 \leq \alpha_i \leq C$ and $0 \leq \alpha_j \leq C$.

Kernel trick: When the optimal Lagrangian multipliers α's have been found by some optimization method then the solution of the equation 3.26 depends only on the inner product $\vec{x}^{(i)T} \vec{x}^{(j)}$. The solution of this inner product may be tedious to calculate when the data vector $\vec{x}$ has a lot of features and dataset is a large. Kernel trick is the way of finding a solution for the inner product without actually solving

the inner product. Thus the equation 3.26 can be rewritten in form of

$$L(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(\vec{x}^{(i)}, \vec{x}^{(j)}) \tag{3.29}$$

where $K()$ is the kernel function. Kernel function can be any function which produces the result that is the result of some inner product. However *valid kernel* is also positive semi-definite [9]. The three most common kernels are:

1. Polynomial kernel: For some $p \in \mathbb{N}^+$

$$K(\vec{x}^{(i)}, \vec{x}^{(j)}) = (1 + \vec{x}^{(i)T} \vec{x}^{(j)})^p \tag{3.30}$$

2. Gaussian kernel: For some $\gamma \in \mathbb{R}^+$

$$K(\vec{x}^{(i)}, \vec{x}^{(j)}) = \exp(-\gamma \|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2) \tag{3.31}$$

3. Perceptron kernel: For some $a, b \in \mathbb{R}^+$

$$K(\vec{x}^{(i)}, \vec{x}^{(j)}) = \tanh(a \vec{x}^{(i)T} \vec{x}^{(j)} - b) \tag{3.32}$$

Another benefit of using kernels is that they will perform a feature space extraction. Feature space extraction makes possible to present a linearly non-separable data in space where it is linearly separable. This is in many cases necessary since the SVM is the binary linear classifier and thus is not able to separate non-linear data correctly. On the other hand the data in practice is usually linearly non-separable.

With feature extraction any linearly non-separable data can be made linearly separable when it is has been extracted to complex enough feature space. For example in case of Polynomial kernel the constant p has a direct affect on the dimensionality and complexity of the feature space.

The effect of feature extraction is illustrated in figure 3.2. Here one dimensional data x has been labelled in two groups. From the left graph of the figure 3.2 can be clearly seen that the data is not linearly separable. When the feature extraction $x \longrightarrow \Phi(x, x^2)$ will be performed from one dimensional space to two dimensional space then the data will become linearly separable in the new feature space Φ.

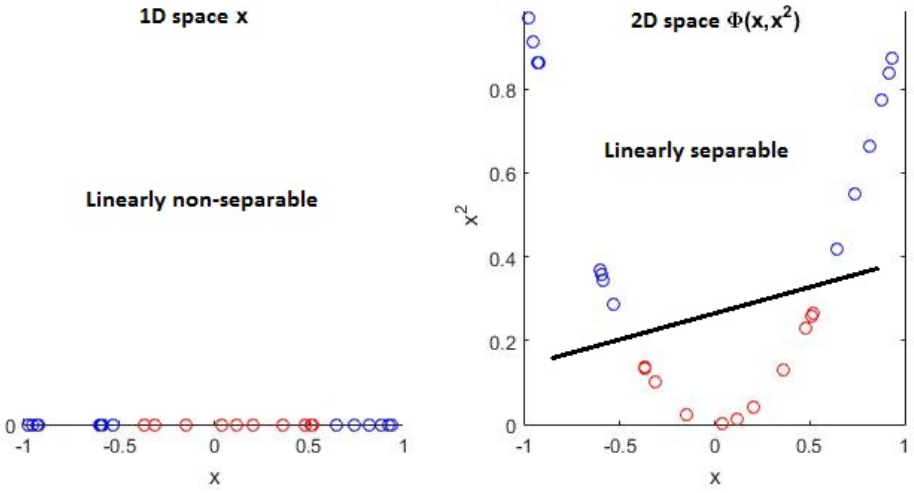


Figure 3.2 *The illustration how feature extraction makes linearly non-separable data to linearly separable.*

Neural Network (NN)

Neural Network (NN), often referred also as an Artificial Neural Network (ANN), is a machine learning method which configuration is strongly inspired by the function of human brain. Human brain can in its simplest form seen as a set of neurons and synapses connecting the neurons. Every neuron is connected by several synapses to several other neurons. Neurons themselves are sort of computational units and synapses are signal transferring units.

The neurons of human brain are distributed in unorganized way and the same same applies with the synaptic connections between the neurons. In NN on the other hand neurons and synapses are organized, in order to have computationally manageable system. The typical configuration of NN is presented in figure 3.3.

As presented in the figure 3.3 neurons are organized in layers. The most common number of layers if three, having a input layer, hidden layer, and output layer. If there is more than three layers in the network then number of hidden layers is increased. If there will be less layers than three in the network then there will be only input and output layer. The network in figure 3.3 has a four layers, thus having two hidden layers.

Typical way of connecting units (input units or neurons) of the network is to connect

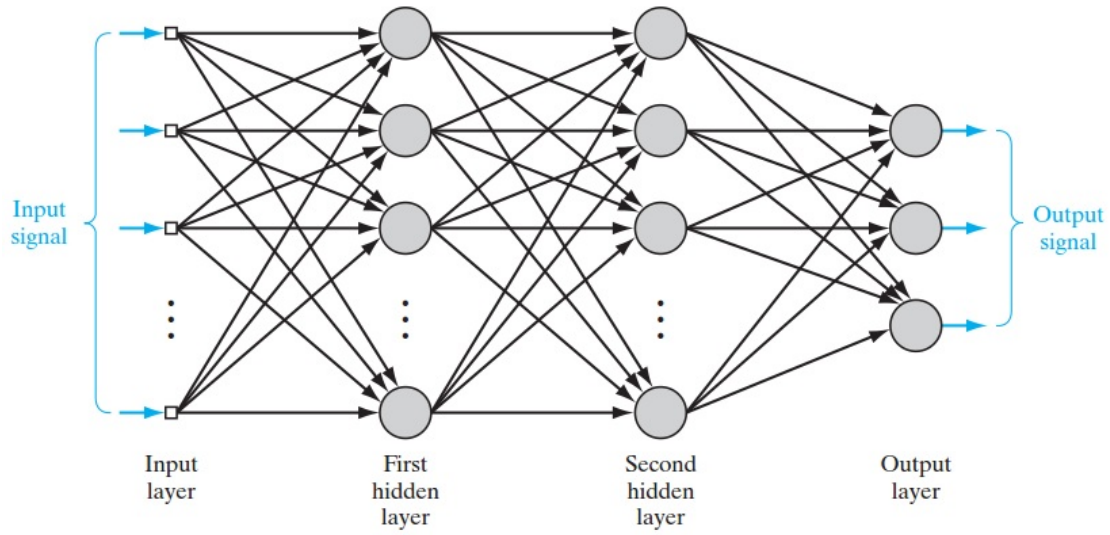


Figure 3.3 *Illustration of multilayer feed-forward NN. Circles are presenting neurons and arrows are presenting connections. [12]*

all units of the previous layer with all units of the next layer. This is also the case in figure 3.3. Network connected like this is called fully connected network. More connections would make the NN computationally challenging.

The function of neuron is illustrated in figure 3.4. Here a neuron k has a input signal x_i, multiplies it by the weight w_{ki}, does the same for all input signals, sums the results, ads biases b_k, have the result v_k out through the Activation function φ and has the output y_k which is further transferred on the next layers or as a final output. This can be summed in one equation as:

$$y_k = \varphi\left(\sum_{i=1}^{m} w_{ki}x_i + b_k\right) \tag{3.33}$$

where m is the number of inputs. By defining $b_k = w_{k0}x_0$ the equation 3.33 can be simplified in form

$$y_k = \varphi\left(\sum_{i=0}^{m} w_{ki}x_i\right) =: \varphi(v_k) \tag{3.34}$$

A good choice is to have $x_0 = 1$.

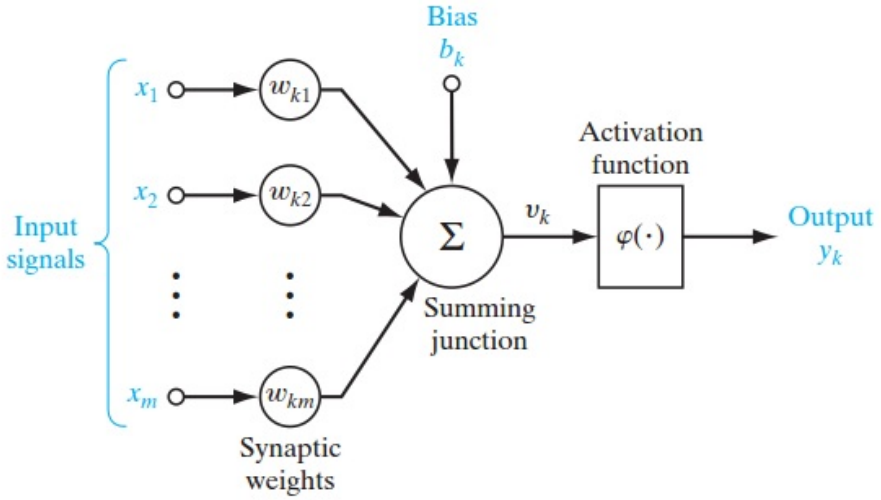


Figure 3.4 *Illustration of the function of the neuron in NN [12]*

Activation function $\varphi()$ can be any function. Practically there are two types of functions commonly used as a activation function:

1. Threshold function:

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0, \\ 0 & \text{if } v < 0, \end{cases} \tag{3.35}$$

2. Sigmoid Function

$$\varphi(v) = \frac{1}{1 + e^{-av}} \tag{3.36}$$

where a is a slope parameter. When $a \to \infty$ then Sigmoid Function acts as a Threshold Function. Here Activation functions can have values between 0 and 1. This is the most common approach. Another common approach is to have Activation function output between the values -1 and 1. The effect of a in Sigmoid Function is demonstrated in figure 3.5.

Learning algorithms There are two main types of learning involved with Neural Networks: supervised learning and unsupervised learning. For example SOM described in section 2.1 is a one of the most powerful and well known form of unsupervised neural networks. In this section the focus is on supervised learning since the aim here is to teach a learning machine to detect the upcoming failure. The training

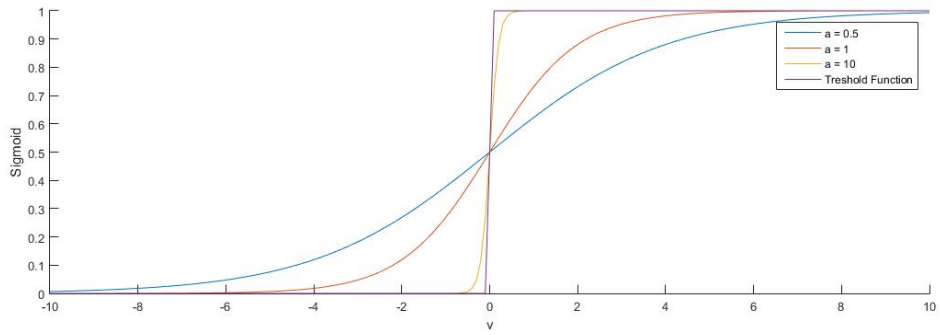


Figure 3.5 *The effect of slope parameter a in Sigmoid Function.*

of supervised NN will be done by Back-Propagation learning algorithm described next.

Back-Propagation: When the input sample n had been feed in to the network then the output neuron j will have an output $y_j(n)$. In supervised learning the knowledge about the desired output of the sample n exists and that is denoted here as $d_j(n)$. Now the error measure can be defined as

$$e_j(n) = d_j(n) - y_j(n) \tag{3.37}$$

We may also define a second error measure that is

$$E_j(n) = \frac{1}{2}(d_j(n) - y_j(n))^2 = \frac{1}{2}e_j^2(n) \tag{3.38}$$

Clearly this $E_j(n)$ is also an error measure and for later purposes it is mathematically convenient.

The total error of the output layer may now defined as

$$E(n) = \sum_{j \in C} E_j(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \tag{3.39}$$

where C holds all the neurons of the layer.

Supervised learning is about adjusting weights of NN. After the data sample n have been feed in to the network the weights need to be readjusted based on the error of

the output. Thus the weights for next epoch $n+1$ will be

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n) \tag{3.40}$$

where

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)} \tag{3.41}$$

where η is learning rate parameter. A constant like 0.2 would be good choice as a learning rate parameter η, but in some cases better convergence of the learning algorithm may be achieved by having $\eta(n)$ as a decreasing function of n [12].

The therm $\partial E(n)/\partial w_{ji}(n)$ of the equation 3.41 can be expressed in following form by using a chain rule:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{3.42}$$

By using the equation 3.39 we get

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \tag{3.43}$$

By using the equation 3.37 we get

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{3.44}$$

By using the equation 3.34 we get

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \tag{3.45}$$

and

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_j(n) \tag{3.46}$$

By substituting the results 3.43, 3.44, 3.45 and 3.46 into the equation 3.42 we get

$$\Delta w_{ji}(n) = -\eta \delta_j(n) y_j(n) \tag{3.47}$$

where

$$\delta_j(n) = e_j(n)\varphi_j^{'}(v_j(n)) \tag{3.48}$$

All the parameters of the equation 3.47 are known when we chose the activation function $\varphi()$ and thus the updated set of the weights for **output layer** can be calculated using equations 3.40 and 3.47.

Now lets define $\Delta w_{kj}(n)$ for the hidden layers. By defining some output of the hidden layer as $y_k(n)$, using an equation 3.41 and chain rule we get

$$\Delta w_{kj}(n) = -\eta \frac{\partial E(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{kj}(n)} \tag{3.49}$$

By using the equation 3.39 we get

$$\frac{\partial E(n)}{\partial y_k(n)} = \sum_j e_j(n) \frac{\partial e_j(n)}{\partial y_k(n)} \tag{3.50}$$

and by chain rule

$$\frac{\partial E(n)}{\partial y_k(n)} = \sum_j e_j(n) \frac{\partial e_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial y_k(n)} \tag{3.51}$$

By using equations 3.34 and 3.37 we get

$$\frac{\partial e_j(n)}{\partial v_j(n)} = -\varphi_j^{'}(v_j(n)) \tag{3.52}$$

Also by using equation 3.34 we get

$$\frac{\partial v_j(n)}{\partial y_k(n)} = w_{jk} \tag{3.53}$$

Now by substituting equations 3.52 and 3.53 into the equation 3.51 we get

$$\frac{\partial E(n)}{\partial y_k(n)} = -\sum_j e_j(n)\varphi_j^{'}(v_j(n)) w_{jk}(n) = -\sum_j \delta_j(n) w_{jk}(n) \tag{3.54}$$

By using equations 3.45, 3.46 and 3.54 we get the equation 3.49 in form of

$$\begin{aligned}\Delta w_{kj}(n) =& \eta\varphi_k^{'}(v_k(n))\sum_j \delta_j(n)w_{jk}(n)y_k(n)\\ =& \eta\delta_k(n)y_k(n)\end{aligned} \tag{3.55}$$

where

$$\delta_k(n) = \varphi_k^{'}(v_k(n))\sum_j \delta_j(n)w_{jk}(n) \tag{3.56}$$

All the parameters of the equation 3.55 are known and thus change in weights $\Delta w_{kj}(n)$ of **hidden layers** can be calculated.

Now the back propagation algorithm can be summed up:

1. Start with the output layer $l = L$ and update the weights of all layers $l = 1..L$ by using equation

$$w_{kj}^{(l)}(n+1) = w_{kj}^{(l)}(n) + \Delta w_{kj}^{(l)}(n) \tag{3.57}$$

where

$$\Delta w_{kj}^{(l)}(n) = \eta\delta_k^{(l)}(n)y_k^{(l-1)}(n) \tag{3.58}$$

where

$$\delta_k^{(l)}(n) = \begin{cases} e_k^{(L)}(n)\varphi_k^{'}(v_k^{(L)}(n)) & \text{for neuron k in output layer L}\\ \varphi_k^{'}(v_k^{(l)}(n))\sum_j \delta_j^{(l+1)}(n)w_{jk}^{(l+1)}(n) & \text{for neuron k in hidden layer l}\end{cases} \tag{3.59}$$

2. Repeat the step 1. until your predefined maximum number of epochs exceed or the error falls below the predefined threshold.

Recurrent Neural Network In Recurrent Neural Network the output signal of the neurons are fed back in to the neurons of the same layer. There exists large number of possibilities of doing the feedback, but some of the most common ways are.

1. Self-feedback: The output signal of neuron is fed back in to the same neuron along with the next data sample or later.

2. No self-feedback: The output signal of neuron is fed as an input of the all other

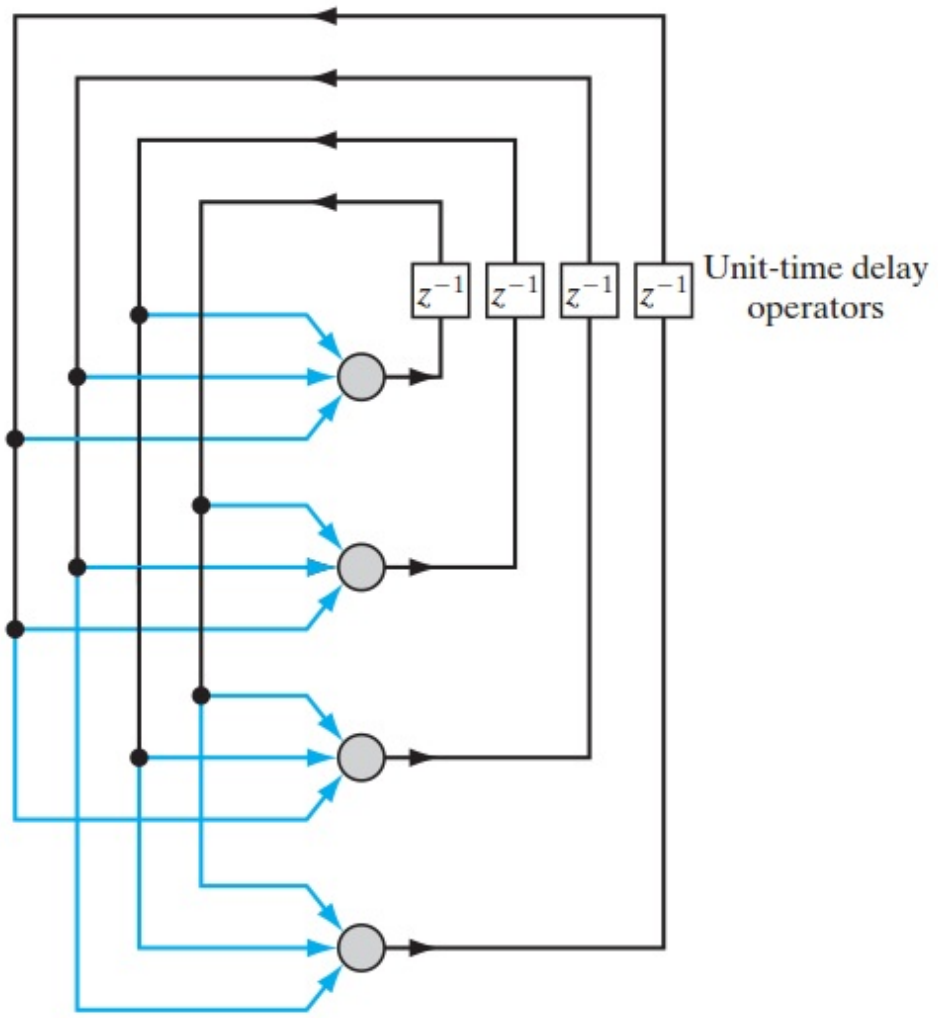


Figure 3.6 *An example of no self-feedback Recurrent Neural Network layer. Figure is form [12].*

neurons of the same layer except the neuron itself along with the next data sample or later. An example of no self-feedback is illustrated in figure 3.6.

3. Full feedback: The output signal of neuron is fed in as an input to the all neurons of the same layer along with the next data sample or later.

3.1.2 Supervised machine generalization capability

Generalization is one of the most important issues involved in machine learning. It is easy to build a complex system which mimics the data with high accuracy. The

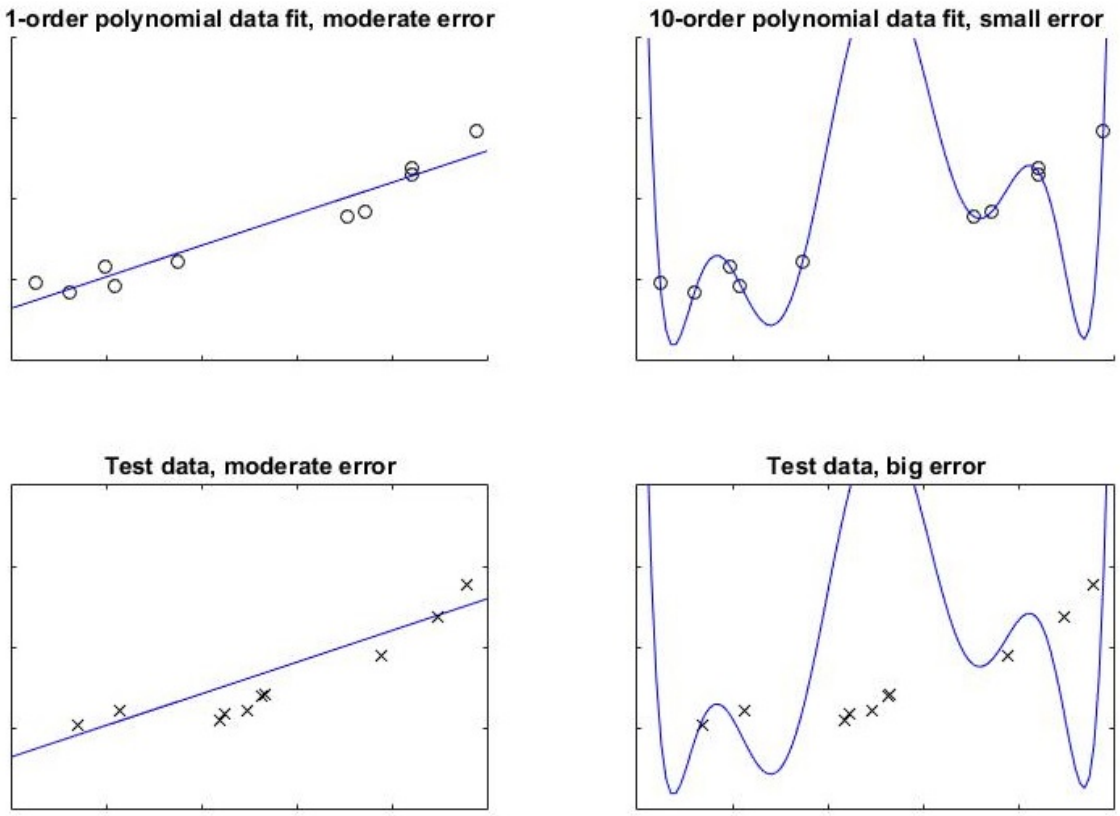


Figure 3.7 *The illustration of over-fitting.*

most complex and accurate system will be the original data itself. The data itself lacks the capability of generalization. Same applies to systems that are too complex. The lack of generalization capability if also called as a *over-fitting*.

The phenomenon of over-fitting is illustrated in figure 3.7. Here 10 samples of data have been presented as o's in the first and second graphs on upper row. In the first graph the first order polynomial fitting have been implemented on the data and in the second graph 10th order polynomial fitting have been implemented on the same data.

The data on the second graph in figure 3.7 is an overfit. Here the fitted curve mimic well the original data but it lack an ability of generalization. When a new data, marked with x's, will be tested on the 10-order polynomial fit then a big error will be generated. With 1-order polynomial fit the moderate error is achieved both in fitting and in testing. Thus here the 1-order fit performs better.

The actual function from which the data samples were generated from in figure 3.7 was 3-order polynomial with some Gaussian noise. Here in the demonstration example the original function was known, but usually when implementing the machine learning methods the underlying function behind the phenomenon is unknown. The Machines ability for generalization will be measured during the training by *validation.*

Validation The idea of validation is following. Split randomly the data available for training into two sets: *training set* and *validation set.* Repeat iteratively:

1. Train the machine by training set and then test the Machine by validation set. Calculate error for validation set. This error is denoted here as E_{out} (out of sample error).

2. Adjust the machine:

- In case of *RBF with k-mean clustering* the number of clusters can be adjusted, basis function can be swapped, free parameters of the basis function can be adjusted, and so on.
- In case of *SVM* the choice of kernel can be adjusted, kernel parameters and especially Box constrain C can be adjusted. Adjusting the Box constraint C has a direct and monotonous effect on generalization.
- In case of *NN* the number of neurons can be adjusted, number of hidden layers can be adjusted and some other features like recurrence of the network can be adjusted.

3. Calculate E_{out}.

Repeat the process iteratively and find the machine that has the smallest E_{out}.

In some cases a little of data is available and in those cases the desire is to use a lot of data for training and spare a little for validation. In this case the procedure called Cross Validation is a good choice.

Cross Validation In Cross Validation the data will be split in M sets. Reasonable way of splitting is to having equal set sizes. The maximum number of M is naturally

the number of samples N in the dataset. The minimum number is 2. Thus $M \in [2, N]$. When $M = N$ then the cross validation is called *N-fold cross validation*. When $M = 10$ then the validation is called *10-fold cross validation*.

The concept of the Cross Validation is following.

1. Split the data into M sets.

2. Exclude one set for later validation and use $M - 1$ sets for training.

3. After training, test the performance of the machine by the one validation set that was excluded from the training.

4. Repeat 2 and 3 but do every time exclude a different set for validation.

The downside of this approach is that you will perform the training and validation M times, which is computationally demanding. The upside of this approach is that all the data will be used both for training and for validation without using the same data for training and validation at same time.

In N-fold validation the validation set is only one sample. Thus the training set will be at size of N-1 and is thus maximum large still to have some data for validation. This approach is suitable for small data-sets. The downsides of N-fold validation is that the training need to be done N times.

The standard way of performing the validation is 10-fold validation [6]. In this approach you will perform the training 10 times, by every time having 90% of data for training and 10% for validation. This approach is the compromise between the computational performance and the size of the training dataset.

VC dimension and VC generalization bound Vapnik-Chervonenkis dimension (VC dimension) gives the maximum number of samples the learner can shatter in the feature space of the sample. *VC generalization bound* is derived from the VC dimension and is a analytical measure for learner giving the upper pound for E_{out} based on measures such complexity of the learner, dimensionality of the data and number of data samples. Thus VC generalization bound describes the learners capability of generalizing.

The concept of VC-dimension is strongly present in almost all machine learning

literature and thus compiled to mention here. In practice VC-dimension is more like conceptual thing and hard to utilize practice with complex systems like NN's. The concept of VC-generalization bound would suit well in situation presented it figure 3.7 where the learner is simple and on the other hand the dataset is small. With small dataset to spare any data for testing would be undesirable and thus the estimate for E_{out} we may have to be done by utilizing VC-generalization bound.

In this study lot of data was available. With the great amount of test data an accurate estimate to E_{out} can be got just by testing the machine, since test error $E_{out} = E_{test}$ is more informative than the upper bond error $E_{out} \leq E_{VC}$.

3.2 Results

3.2.1 Generalization capability testing

During the training the performance of the machine is validated as described in section 3.1.2. In mathematical point of view the validation data has not been used directly for training. On the other-hand in philosophical point of view the validation dataset have been also involved for training, since the choice of the final Machine configuration has been affected by the validation data. In order to measure the absolute performance of the machine, testing is needed.

The idea of testing is to test the performance of the machine by using the data that has not been used in any way for final chose of the machine. This data has not been used for training, for validation or either for making the choice in our intuitive mind for which machine (RBF, SVM, NN, ...) to use in the first place [6].

Here the testing has been performed by configuring six Lessons.

- Lesson 1: the **healthy flights of Aeroplane A** have been totally excluded from the training and validation. From the rest of the flights the data have been separated for training and testing as described in table 3.1.
- Lesson 2: the **healthy flight of Aeroplane B** have been totally excluded from the training and validation. From the rest of the flights the data have been separated for training and testing as described in table 3.1.

- Lesson 3: the **healthy flight of Aeroplane C** have been totally excluded from the training and validation. From the rest of the flights the data have been separated for training and testing as described in table 3.1.
- Lesson 4: the **healthy flight of Aeroplane D** have been totally excluded from the training and validation. From the rest of the flights the data have been separated for training and testing as described in table 3.1.
- Lesson 5: the **first failure flight of Aeroplane A** have been totally excluded from the training and validation. From the rest of the flights the data have been separated for training and testing as described in table 3.1.
- Lesson 6: the **second failure flight of Aeroplane A** have been totally excluded from the training and validation. From the rest of the flights the data have been separated for training and testing as described in table 3.1.

Table 3.1 *Data usage in % for Training and Testing in RBF, SVM and NN.*

	Training / Testing		
	RBF	SVM	NN
Aeroplane A healthy flights	10/90	2/98	10/90
Aeroplane B healthy flight	18/82	4/96	18/82
Aeroplane C healthy flight	27/73	5/95	27/73
Aeroplane D healthy flight	50/50	46/54	50/50
Aeroplane A failure flight I	50/50	13/87	50/50
Aeroplane A failure flight II	50/50	20/80	50/50
Total	44/66	15/85	44/66

Since in this specific case a lot of data was available the most of the data was used for testing. In case of RBF and NN 66 % of the data was used for testing. In case of SVM 85 % of the data was used for testing. The difference of of data separation in training/testing between the methods (RBF, SVM, NN) is affected by the computation speed; that is all machines were given approximately equal time on CPU.

In practice a big testing dataset confirms well the performance of the machine. On the other hand better performance of he machine might have been received with bigger training data set. Thus the compromise needed to be done between the two extremes: having theoretically accurately performing machine which performance is not tested or having poorly performing machine which performance is well tested.

Here the compromise have been strongly affected by the computational resources. Also here the aim is to compare Machines and test the potential of some learning machines, not to have final optimized machine for practical use.

The results of testing each machine will be summarized in tables similar like 3.2. The table 3.2 is an hypothetical and gives the idea what short of results the perfectly working hypothetical learning machine would produce. The intuition behind the table is that it would be preferable that the machine would not see any (0 %) FIHDS's in the data of the flights which did never failed and see all the data (100 %) as a FIHDS's for the flights which did end up with the failure.

Table 3.2 *The percentage of testing data seen as a FIHDS's by the hypothetical perfect learning machine.*

System runs	Lesson 1	Lesson 2	Lesson 3	Lesson 4	Lesson 5	Lesson 6
A healthy(40)	0	0	0	0	0	0
B healthy(1)	0	0	0	0	0	0
C healthy(1)	0	0	0	0	0	0
D healthy(1)	0	0	0	0	0	0
A fail I(1)	100	100	100	100	100	100
A fail II(1)	100	100	100	100	100	100

3.2.2 Radial Basis Function (RBF) with K-mean clustering

For Radial Basis Function there exists several options available for basis function as it was pointed out in section 3.1.1. Here the Gaussian function was chosen:

$$\varphi(\|x - x_n\|) = exp(-\lambda\|x - x_n\|^2) \tag{3.60}$$

,where $\lambda > 0$. The reason for the choice is that the Gaussian function is treated as a standard in many literature and often no other basis functions are not even introduced.

In order to come out with the solution the first thing to do is to have the decision about the number of clusters K to be used use. In limited range the out of sample error E_{out} is decreasing as a function of the increasing number of clusters. The small test presented in figure 3.8 supports the statement.

Here in the test runs the number of clusters have been chosen to be ten. The choice

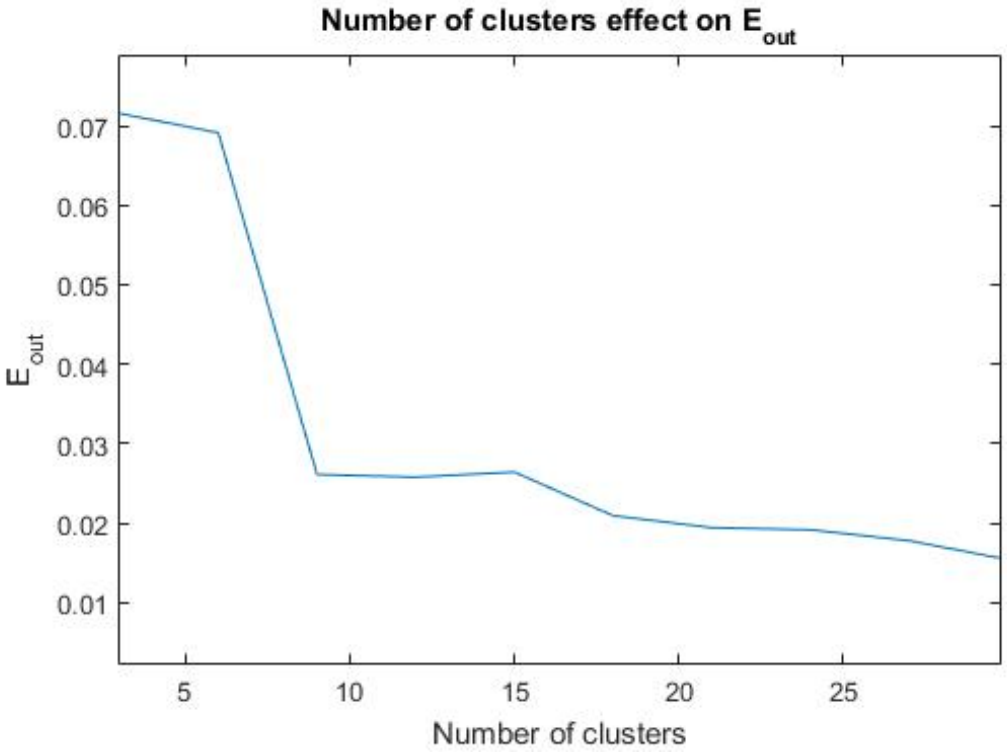


Figure 3.8 *The effect of cluster center number on out of sample error of trained RBF.*

was compromise between the computation speed and the performance, but mostly emphasising the computation speed.

Since here the Gaussian function was the choose to be used it leads to the one free parameter to be able to adjust, that is λ. With the datasets of this study the out of sample error E_{out} behaved respect to the λ as presented in the figure 3.9. Based on this result the λ have been fixed to be 1.3 for further calculations.

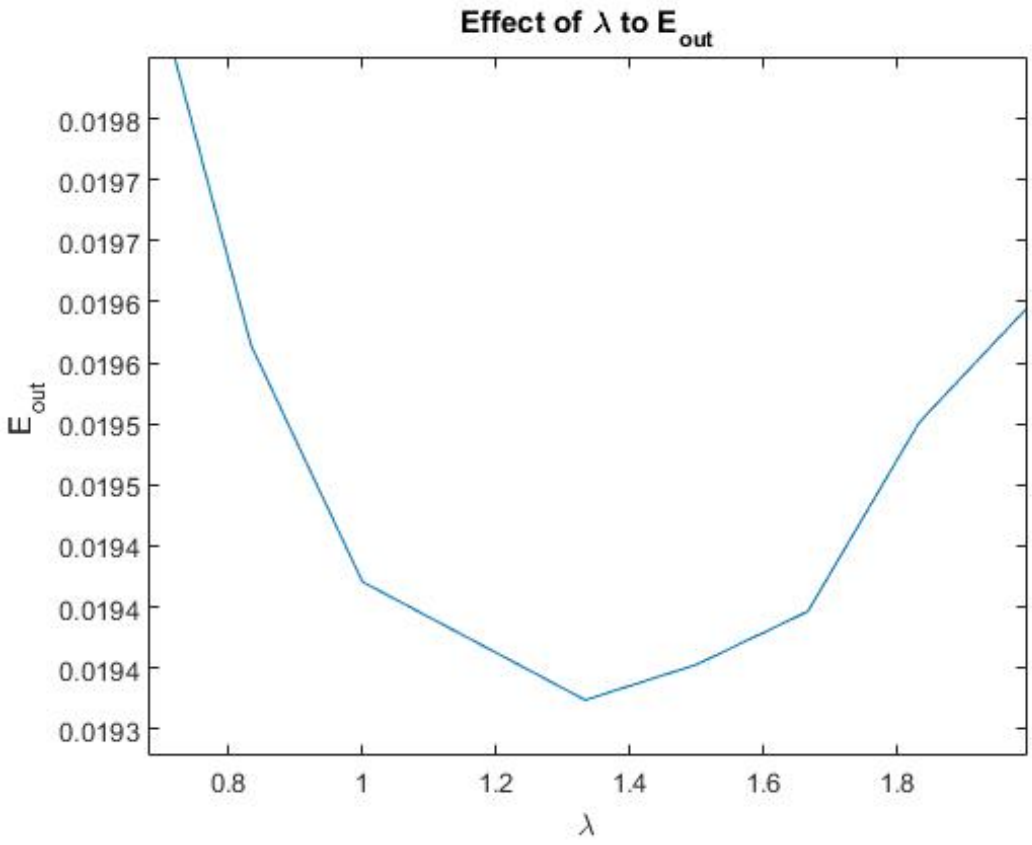


Figure 3.9 *The effect of λ on out of sample error.*

The actual calculation of RBF was done by using the equations described in the section 3.1.1. The clusters of K-mean clustering were calculated by using Matlab inbuilt function *kmeans* [3].

The average results for training ten separate RBF's with K-means have been presented in table 3.3.

Table 3.3 *Percentage of testing data seen as a FIHDS's by the trained RBF. ± indicates the standard deviation of ten separate runs. The datasets with ∗ have been totally excluded from the training.*

System runs	Lesson 1	Lesson 2	Lesson 3	Lesson 4	Lesson 5	Lesson 6
A healthy(40)	$*98 \pm 2$	1 ± 0	1 ± 0	1 ± 0	1 ± 0	0 ± 0
B healthy(1)	1 ± 3	$*98 \pm 1$	0 ± 0	1 ± 1	2 ± 2	0 ± 0
C healthy(1)	1 ± 0	3 ± 0	$*33 \pm 36$	2 ± 1	1 ± 0	0 ± 0
D healthy(1)	98 ± 3	100 ± 0	91 ± 2	$*92 \pm 1$	2 ± 1	88 ± 1
A fail I(1)	94 ± 2	97 ± 1	95 ± 2	100 ± 0	$*44 \pm 32$	96 ± 1
A fail II(1)	91 ± 8	99 ± 0	90 ± 10	95 ± 3	66 ± 38	$*54 \pm 21$

3.2.3 Support Vector Machine (SVM)

SVM was trained here by using the Matlab's inbuilt function *fitcsvm* [1]. As a kernel Gaussian kernel described in section 3.1.1 was used. The results are listed in the table 3.4.

Table 3.4 *Percentage of testing data seen as a FIHDS's by the trained SVM. ± indicates the standard deviation of ten separate runs. The datasets with ∗ have been totally excluded from the training.*

System runs	Lesson 1	Lesson 2	Lesson 3	Lesson 4	Lesson 5	Lesson 6
A healthy(40)	$*0 \pm 0$	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
B healthy(1)	0 ± 0	$*0 \pm 0$	0 ± 0	0 ± 0	0 ± 0	0 ± 0
C healthy(1)	0 ± 0	0 ± 0	$*3 \pm 3$	0 ± 0	0 ± 0	0 ± 0
D healthy(1)	3 ± 0	3 ± 0	3 ± 0	$*93 \pm 0$	2 ± 0	1 ± 0
A fail I(1)	96 ± 0	96 ± 0	97 ± 0	99 ± 0	$*63 \pm 8$	95 ± 0
A fail II(1)	98 ± 0	98 ± 1	98 ± 0	99 ± 0	97 ± 0	$*44 \pm 2$

As for the Box Constraint $C = 1$ was used here. The reason is partly illustrated in figure 3.10. From the figure it can be seen that when $C \geq 1$ the error dos not vary much in this specific case. On the other hand increase in the box constraint can lead to longer training times [1], and thus $C = 1$ is here a compromise between low error and reasonable computation time.

During the all Lessons of the table 3.4 when the SVM was trained then the average number of Support Vectors out of the hole number of samples used for training was 18% with the standard deviation of 2%. In general if all data sample (100%)

Figure 3.10 *The effect of box constraint on error.*

presented to SVM will come out as as support vectors then the machine would be a really bad generalizer and only good memorizer. The other extreme would be to have only two ($\sim 0\%$) support vectors, one of each side of two classes then the machine would be a great generalizer but not having much of the original wisdom of the data. The optimum lies between these two cases and is case-specific.

3.2.4 Neural Networks (NN)

The the functions of Neural Network Toolbox of Matlab [4] were used to build a several neural network configurations in order to classify FIHDS's out of the healthy data. The effect of following features of NN on final result was investigated:

1. Learning algorithms.

2. Number of neurons in layer.

3. Number of layers.

4. Feedforward and Feedback (time delay in Matlab).

Effect of learning algorithms The most significant effect on final result had the choose of learning algorithm. All nine inbuilt back propagation learning algorithms of Matlab (Levenberg-Marquardt, BFGS Quasi-Newton, Resilient Backpropagation, Scaled Conjugate Gradient, Conjugate Gradient with Powell/Beale Restarts, Fletcher-Powell Conjugate Gradient, Polak-Ribiére Conjugate Gradient,

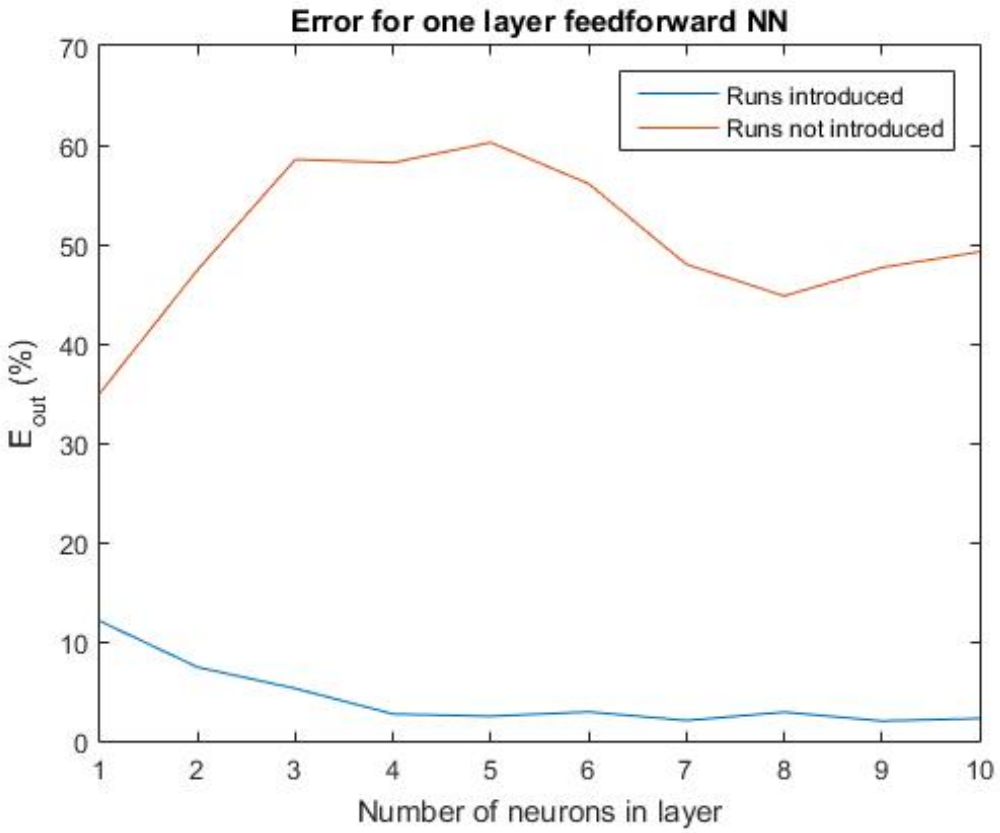


Figure 3.11 *The effect of number of neurons in one hidden layer Feedforward Neural Network on* E_{out}.

One Step Secant, Variable Learning Rate Back-propagation) were tested and only one (Levenberg-Marquardt) could provide a reasonable solution for the problem in issue.

Effect of layer size The effect of layer size on the performance of the NN in illustrated in figure 3.11. As the layer size increases the E_{out} decreases. The E_{out} reaches its minimum already with four neurons in single layer and does not significantly decrease with bigger layer sizes.

Theoretically the increasing size of layer should in some point start to cause the overfitting, discussed in the section 3.1.2, and thus cause increase in E_{out}. This phenomenon was not detected here, but on the other hand in this case the Matlab NN Toolbox internal cross-validation algorithm and early stopping prevents the overfitting.

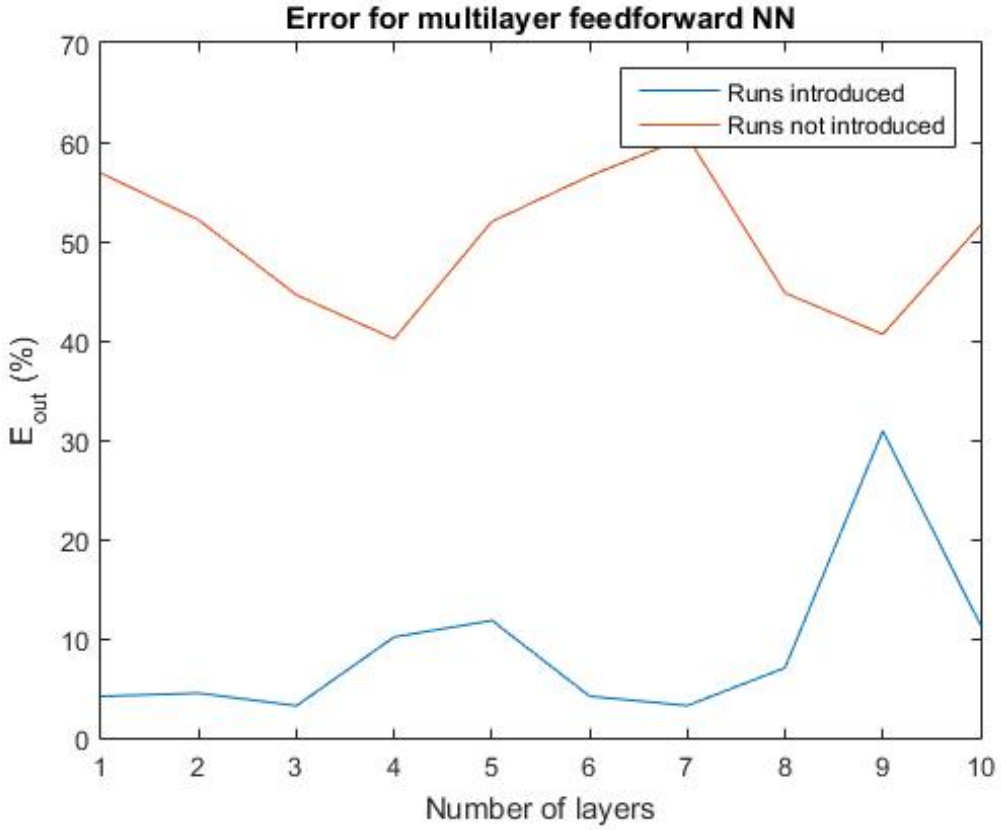


Figure 3.12 *The effect of number of hidden layers in Feedforward Neural Network on E_{out}. One layer contains four neurons.*

Since the increase in layer size causes the increase in computational times the conclusion here based on the figure 3.11 was that the optimum number of neuron in this specific case would be four.

Effect of number of hidden layers The increase of hidden layer number here had no improving effect on NN performance. This is illustrated in figure 3.12 where several combinations of four neuron layers were constructed and the E_{out} was calculated. In practice increasing number of layers seems to make NN performance worse. This phenomenon can be explained by the overfitting, discussed in the section 3.1.2.

Effect of feedback The effect of feedback, that is time delay in Matlab toolbox, on the performance of this specific NN was tested. Feedback property in NN here

destroyed the performance of the NN and no reasonable results was received. The result was the same with several variations of time delay units from 1 to 10. This indicates that the underlying process is not dynamic [10]. In practise the operation of flight control surfaces is probably dynamic, but probably not in the scale of our data measurement intervals.

Here based on the four points presented above the NN of table 3.5 was used for FIHDS's classification:

Table 3.5 *The NN used for classifying FIHDS's out of new data*

Learning algorithm	Levenberg-Marquardt
Number of hidden layers	1
Number of neurons in hidden layer	4
Connections	Fully connected Feedforward

The criteria for choosing this network was its good performance and light computational demand. The network produced the following result presented in the table 3.6.

Table 3.6 *Percentage of testing data seen as a FIHDS's by the trained NN. ± indicates the standard deviation of ten separate runs. The datasets with ∗ have been totally excluded from the training.*

System runs	Lesson 1	Lesson 2	Lesson 3	Lesson 4	Lesson 5	Lesson 6
A healthy(40)	$*98 \pm 2$	0 ± 0	0 ± 1	0 ± 0	0 ± 0	0 ± 0
B healthy(1)	0 ± 0	$*1 \pm 3$	0 ± 0	0 ± 0	0 ± 0	0 ± 0
C healthy(1)	0 ± 0	1 ± 1	$*47 \pm 37$	0 ± 0	0 ± 0	0 ± 0
D healthy(1)	17 ± 8	16 ± 6	19 ± 9	$*93 \pm 1$	5 ± 1	12 ± 2
A fail I(1)	96 ± 3	96 ± 2	95 ± 4	100 ± 0	$*64 \pm 15$	96 ± 1
A fail II(1)	97 ± 3	98 ± 2	98 ± 1	98 ± 2	97 ± 2	$*44 \pm 10$

4. CONCLUSIONS

The aim of this study was to test some commonly known and promising machine learning methods like SOM, RBF, SVM, NN and K-mean clustering for predicting the failure of aeroplane flight control surface. The recorded data from Aeroplane A was available for the task, including several flights from several years. Also some small amount of data from other similar aeroplanes was available (see table 1.1).

The study was segmented into two parts. In the first part (Part 1) the task was to decide what to teach to machine. For this reason Failure Indicating Healthy Data Sample (FIHDS) was defined. The existence of FIHDS in healthy historical data was examined with three different methods: Self Organising Maps (SOM), K-mean clustering and in heuristic way. From the flight data of Aeroplane A the FIHDS's were found with all of the methods.

The second task was to teach the learning machine to detect the FIHDS's from new data. This issue was covered in Part 2. For this purpose the data was divide in to two parts: one part of the data was used to to teach the machine and the other part of the data was used to later test the performance of the trained machine.

Three learning machines were used for this purpose: Radial Basis Function (RBF), Support Vector Machine (SVM) and Neural Network (NN). The machines were first trained and then tested. The data used for training was not used for testing and vice versa.

In order to test the ultimate performance of the machines six lessons were constructed. In each lesson one dataset of flight(s) were totally excluded from the training. For rest of the flights the data usage for training is described in table 3.1. The testing results of the lessons are summarized into the tables 3.3, 3.4 and 3.6.

As for comparison table 3.2 presents hypothetical perfect results. The intuition behind this table is that when the run will not end with failure it is desirable that

the learning machine would not see any FIHDS's in the data and when run will end with failure it is desirable that the learning machine will see a lot of FIHDS's in the data.

The Lesson 6 in tables 3.3, 3.4 and 3.6 mimic the actual historical process of the Aeroplane A. In this lesson we consider ourselves in the moment where Aeroplane A has been failed once but not yet at second time. At this point we hypothetically build a machines to detect FIHDS's. We train the machines with some data from the healthy flights of Aeroplane A, healthy flight data's of some other similar aeroplanes and the data from *Aeroplane A fail run I*, that is the only failure run of Aeroplane A which we have at this moment. Then the Aeroplane A will be fixed. Then we start fly with the Aeroplane A again and detect on-line the data of the Aeroplane A with our machines. Our machines performed following: RBF sees 54 % data coming out uniformly as a FIHDS's, SVM sees 44 % data coming out uniformly as a FIHDS's and NN sees 44 % data coming out uniformly as a FIHDS's. This level of FIHDS's should alarm about the upcoming failure since normally when system is healthy the fraction of FIHDS's out of all data is close to zero (see tables 3.3, 3.4 and 3.6). It practice we know that *Aeroplane A failure flight II* ends with failure. Thus in this specific case all the machines showed a great potential for purpose of predict the upcoming failure.

From the tables 3.3, 3.4 and 3.6 it can be seen that the learning machines are classifying the *Aeroplane D healthy flight* in relatively high frequency (especially RBF) as a FIHDS's even the flight did not ended with failure in practice. The reason for this is probably the fact that there was relatively small amount of data available from Aeroplane D flight. Thus the dataset used for teaching the machines was also small. The size of the data set was ten times smaller than the sizes of the dataset s of Aeroplane B and Aeroplane C and over one hundred times smaller that the dataset of Aeroplane A.

The results in diagonals in tables 3.3, 3.4 and 3.6 are not so good as the rest of the results. This indicates that when the data of some system is totally excluded from training the machine then the machine is poorly capable to predict the behaviour of this particular system. The same was concluded by Pascual D. G. [16] stating that "black-box approach...cannot identify situations that have not previously occurred."

When comparing the table 3.2 with the tables 3.3, 3.4 and 3.6 it can be concluded that SVM performed best out of the three Machines. SVM did not see a lot of

FIHDS's in the data of healthy flights thus having a low false alarm rate. On the other hand SVM did see a prominent amounts of FIHDS's in the data of the failure flights and most importantly in the unfamiliar sets (marked with *). The only clear misclassification of SVM was Aeroplane D healthy flight in Lesson 4. On the other hand this is forgiveable since during the Lesson 4 no data from the Aeroplane D was used to train the learning machine.

Besides that SVM performed the best there are exists some other advantage with SVM. For example here the SVM case was capable to achieve a great performance with relatively small training data (see table 3.1). This fact indicates that there exist even further improvement potential with SVM. The fact also indicated that the results of SVM are the most reliable since a greatest fraction of the data were used for testing. The downside with the SVM is that with the big datasets the algorithm is computationally demanding.

The second best performance was received with NN. The advantage with NN is that it can be easy further trained with new data. Thus it would fit well in on-line learning and monitoring. The downsides with the NN is that the optimal configuration is hard to find.

The third best performance was received with RBF. Still there exists a great potential for improvement with RBF in the context of this application, since the RBF here was coded by ourselves and the optimization of its parameters was not that systematic when comparing with the SVM Matlab functions and NN Matlab Toolbox functions.

All three machines did see FIHDS's in uniform frequency during the Aeroplane A failure flight II. Thus the indication about the upcoming failure did exist at from the beginning of the flight. The real time between the beginning of the flight and failure was several tens of minutes. This fact can be interpret that these machines would have potential for forecasting the aeroplane flight control surface failures.

5. DISCUSSION

Here the feasibility of some common machine learning algorithms for predicting the upcoming failure of aeroplane flight control surface was examined. The algorithms discussed here were SOM, SVM, NN, RBF and K-mean clustering. These algorithms are just a small set of the all machine learning algorithms available. These specific algorithms were chosen here based on their promising results from the other studies and their applicability for solving the issue in hand. Also the availability of ready build packages and functions for Matlab was strongly affecting to the choice. Thus this study can be further extended by having more algorithms included. Here these few algorithms showed a promising results.

The optimum solution for the specific issue here was not aim to find. For example RBF can be further improved by having more K-clusters. Here only 10 clusters was used even the data here was in scale of millions data sample. RBF can be also improved by the systematic choose of basis function and the free parameters of the basis function.

Also SVM have a potential for further improvement for example by having greater fraction of data for training. This could be done for example by calculating several sets of support vectors and then finding the support vectors of the support vectors and so on. Another possibility for improving SVM would lie in the systematic choice of kernel and the adjustment of kernel parameters. Here the standard Gaussian kernel was used. There exists a lot of kernel options to be tested and some of them are mentioned in section 3.1.1.

Neural network has a lot of adjustable parameters: learning algorithms, number of neurons in layer, number of layers, feedback feature and so on. In practise there exists a tremendous amount of options how to configure NN. Here systematic configure $\rightarrow$ train $\rightarrow$ test $\rightarrow$ compare $\rightarrow$ configure $\rightarrow$ train $\rightarrow$ test $\rightarrow$ compare $\rightarrow$ ... would have lead bather results.

Here a straightforward testing on learning algorithms were done and based on that it seems that the SVM has a great potential for further improvement and practical use. Without a careful adjustments the SVM still performed superb and the systematic adjustment would have been make it to perform even better. Thus we suggest that in this sort of similar issues to use SVM and further develop it. Special focus should be on the choose of kernels, creating new kernels and adjusting the kernel parameters.

In this study the learning machines showed a great potential for finding anomalies from the historical data and further classifying them from the new data. The machines capability of detecting FIHDS's was well tested here. On the other hand the statement: "FIHDS's defined in our way will really forecast in practice the failure of flight control surface", need to be further examined. For this purpose more datasets are needed from several aeroplanes including chronologically: healthy flights, failure flight(s), healthy flights.

The source data of this study was from the aeroplanes of Finnish Air Forces and the results achieved from this data by machine learning methods were promising. On the other hand we believe that the methods presented here for failure forecasting would be applicable for any system with data. The modern learning machines seem to be powerful for fitting any arbitrary complicated function and the problem with fits are rather over-fits that under-fits. The bottlenecks with the methods presented here are in practise the availability and the quality of the data.

If the data of the Finnish Air Forces will develop over the time in such way that the failure data classes will be more presented then more accurate supervised learning machines could be developed and implemented for the Flight Control Surface failure prediction. The results from the methods could be further used in practice to avoid some potential safety risks due to the maintenance decision support they could provide. Also similar systems may be built for other subsystems of the aeroplanes or for entirely new systems.

In future the development of the methods needs to continue in order to achieve practically applicable results for Finnish Air Forces. Supervised learning machines presented here are all needing the data of failures besides normal data. This is a problematic since waiting failure data sets means the same as waiting failures. This is not a desired scenario and thus the study needs to focus in the future more on the methods which are not requiring the failure data. These methods are unsupervised learning machines from which some were presented here and sowed promising results.

However further development with unsupervised learning machines are needed in order to have easily interpretable results out of them for maintenance support.

In order to have practical advantage in future from these results for Finnish Defence Forces, more work and cooperation need to be done. One scenario about how the process should proceed includes the following steps.

1. Verifying the fact that the model of this study can really predict the failures of Flight Control Surfaces. This step requires the data from other aeroplanes with similar failures. Data sets should include here chronologically ...healthy flight, healthy flight, **failure flight**, healthy flight...

2. Building the SVM and NN for practical failure forecasting. This step requires data from all the aeroplanes which will be monitored in future and the data from both healthy flights and failure flights is needed. Computer program including SVM and NN will run on Finnish Air Forces servers, analysing the new flight data sets and reporting the results.

3. Developing unsupervised learning methods and they interpretation routines in such way that also unsupervised learning machines could be used directly for failure forecasting. This is a huge field of study but when succeeded in practice it will relieve us from the need of failure data which will be remarkable advantage.

BIBLIOGRAPHY

[1] fitcsvm. [Online]. Available: http://se.mathworks.com/help/stats/fitcsvm.html

[2] Flight control surfaces. [Online]. Available: https://en.wikipedia.org/wiki/Flight_control_surfaces

[3] kmeans. [Online]. Available: http://se.mathworks.com/help/stats/kmeans.html

[4] Neural network toolbox. [Online]. Available: http://se.mathworks.com/products/neural-network/

[5] (2015) Som toolbox. [Online]. Available: http://www.cis.hut.fi/somtoolbox/

[6] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data.* [United States]: AMLBook.com, cop. 2012.

[7] D. P. Bertsekas, *Nonlinear programming.* Belmont (MA): Athena Scientific, 1995.

[8] M. Cottrell, P. Gaubert, C. Eloy, D. FranÃ§ois, G. Hallaux, J. Lacaille, and M. Verleysen, *Fault prediction in aircraft engines using self-organizing maps*, ser. Advances in Self-Organizing Maps. Springer, 2009, pp. 37–44.

[9] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*, repr. with corr ed. Cambridge: Cambridge University Press, 2000.

[10] G. Dreyfus, *Neural networks: methodology and applications.* Springer Science & Business Media, 2005.

[11] D. Gluss and E. W. Weisstein. Lagrange multiplier. [Online]. Available: http://mathworld.wolfram.com/LagrangeMultiplier.html

[12] S. S. Haykin, *Neural networks and learning machines.* Pearson Education Upper Saddle River, 2009, vol. 3.

[13] R. Huang, L. Xi, X. Li, C. R. Liu, H. Qiu, and J. Lee, "Residual life predictions for ball bearings based on self-organizing map and back propagation neural

network methods," *Mechanical Systems and Signal Processing*, vol. 21, no. 1, pp. 193–207, 2007.

[14] N. Karunanithi, D. Whitley, and Y. K. Malaiya, "Using neural networks in reliability prediction," *Software, IEEE*, vol. 9, no. 4, pp. 53–59, 1992, iD: 1.

[15] T. Y. Kim, K. J. Oh, I. Sohn, and C. Hwang, "Usefulness of artificial neural networks for early warning system of economic crisis," *Expert Systems with Applications*, vol. 26, no. 4, pp. 583–590, 2004.

[16] D. G. Pascual, *Artificial Intelligence Tools: Decision Support Systems in Condition Monitoring and Diagnosis*. CRC Press, 2015.

[17] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys (CSUR)*, vol. 42, no. 3, p. 10, 2010.

[18] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas, "som toolbox for matlab 5. report a57, helsinki university of technology," *Neural Networks Research Centre, Espoo, Finland*, 2000.

[19] R. Yam, P. Tse, L. Li, and P. Tu, "Intelligent predictive decision support system for condition-based maintenance," *The International Journal of Advanced Manufacturing Technology*, vol. 17, no. 5, pp. 383–391, 2001.

APPENDIX A: SUMMATION OF THE RESULTS

Table 1 *Percentage of testing data seen as a FIHDS's by the trained RBF.*

System runs	Lesson 1	Lesson 2	Lesson 3	Lesson 4	Lesson 5	Lesson 6
A healthy(40)	$*98 \pm 2$	1 ± 0	1 ± 0	1 ± 0	1 ± 0	0 ± 0
B healthy(1)	1 ± 3	$*98 \pm 1$	0 ± 0	1 ± 1	2 ± 2	0 ± 0
C healthy(1)	1 ± 0	3 ± 0	$*33 \pm 36$	2 ± 1	1 ± 0	0 ± 0
D healthy(1)	98 ± 3	100 ± 0	91 ± 2	$*92 \pm 1$	2 ± 1	88 ± 1
A fail I(1)	94 ± 2	97 ± 1	95 ± 2	100 ± 0	$*44 \pm 32$	96 ± 1
A fail II(1)	91 ± 8	99 ± 0	90 ± 10	95 ± 3	66 ± 38	$*54 \pm 21$

Table 2 *Percentage of testing data seen as a FIHDS's by the trained SVM.*

System runs	Lesson 1	Lesson 2	Lesson 3	Lesson 4	Lesson 5	Lesson 6
A hellthy(40)	$*0 \pm 0$	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
B heatlhy(1)	0 ± 0	$*0 \pm 0$	0 ± 0	0 ± 0	0 ± 0	0 ± 0
C healthy(1)	0 ± 0	0 ± 0	$*3 \pm 3$	0 ± 0	0 ± 0	0 ± 0
D healthy(1)	3 ± 0	3 ± 0	3 ± 0	$*93 \pm 0$	2 ± 0	1 ± 0
A fail I(1)	96 ± 0	96 ± 0	97 ± 0	99 ± 0	$*63 \pm 8$	95 ± 0
A fail II(1)	98 ± 0	98 ± 1	98 ± 0	99 ± 0	97 ± 0	$*44 \pm 2$

Table 3 *Percentage of testing data seen as a FIHDS's by the trained NN.*

System runs	Lesson 1	Lesson 2	Lesson 3	Lesson 4	Lesson 5	Lesson 6
A healthy(40)	$*98 \pm 2$	0 ± 0	0 ± 1	0 ± 0	0 ± 0	0 ± 0
B healthy(1)	0 ± 0	$*1 \pm 3$	0 ± 0	0 ± 0	0 ± 0	0 ± 0
C healthy(1)	0 ± 0	1 ± 1	$*47 \pm 37$	0 ± 0	0 ± 0	0 ± 0
D healthy(1)	17 ± 8	16 ± 6	19 ± 9	$*93 \pm 1$	5 ± 1	12 ± 2
A fail I(1)	96 ± 3	96 ± 2	95 ± 4	100 ± 0	$*64 \pm 15$	96 ± 1
A fail II(1)	97 ± 3	98 ± 2	98 ± 1	98 ± 2	97 ± 2	$*44 \pm 10$

Datasets with * excluded (100 %) from the training.

Table 4 *Desired optimal result for percentage of testing data seen as a FIHDS's.*

System runs	Lesson 1	Lesson 2	Lesson 3	Lesson 4	Lesson 5	Lesson 6
A healthy(40)	0	0	0	0	0	0
B healthy(1)	0	0	0	0	0	0
C healthy(1)	0	0	0	0	0	0
D healthy(1)	0	0	0	0	0	0
A fail I(1)	100	100	100	100	100	100
A fail II(1)	100	100	100	100	100	100